intel®

APPLICATION NOTE

AP-145

June 1982

Synchronous Communication with the 8274 Multiple Protocol Serial Controller

Sikandar Naqvi
Application Engineer

ORDER NUMBER: 210403-001

INTRODUCTION:

The INTEL 8274 is a Multi-Protocol Serial Controller, capable of handling both asynchronous and synchronous communication protocols. Its programmable features allow it to be configured in various operating modes, providing optimization to given data communication application.

This application note describes the features of the MPSC in Synchronous Communication applications only. It is strongly recommended that the reader read the 8274 Data Sheet and Application Note AP134 "Asynchronous Communication with the 8274 Multi-Protocol Serial Controller" before reading this Application Note. This Application note assumes that the reader is familiar with the basic structure of the MPSC, in terms of pin description, Read/Write registers and asynchronous communication with the 8274. Appendix A contains the software listings of the Application Example and Appendix B shows the MPSC Read/Write Registers for quick reference.

The first section of this application note presents an overview of the various sysnchronous protocols. The second section discusses the block diagram description of the MPSC. This is followed by the description of MPSC interrupt structure and mode of operation in the third and fourth sections. The fifth section describes a hardware/software example, using the INTEL single board computer iSBC88/45 as the hardware vehicle. The sixth section consists of some specialized applications of the MPSC. Finally, in section seven, some useful programming hints are summarized.

OPENING FLAG BYTE	ADDRESS* FIELD(A)	CONTROL** FIELD(C)	DATA FIELD	FRAME CHECK SEQUENCE	CLOSING FLAG BYTE

Figure 1. HDLC/SDLC Frame Format

* Extendable to 2 or More Bytes
** Extendable to 2 Bytes

SYNCHRONOUS PROTOCOL OVERVIEW

This section presents an overview of various synchronous protocols. The contents of this section are fairly tutorial and may be skipped by the more knowledgeable reader.

Bit Oriented Protocols Overview

Bit oriented protocols have been defined to manage the flow of information on data communication links. One of the most widely known protocol is the one defined by the International Standards Organization: HDLC (High Level Data Link Control). The American Standard Associations' protocol, ADCCP is similar to HDLC. CCITT Recommendation X.25 layer 2 is also an acceptable version of HDLC. Finally, IBM's SDLC (Synchoronous Data Link Control) is also a subset of the HDLC.

In this section, we will concentrate most of our discussion on HDLC. Figure 1 shows a basic HDLC frame format.

A frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags - opening and closing flags. An address field is 8 bits wide, extendable to 2 or more bytes. The control field is also 8 bits wide, extendable to two bytes. The data field or information field may be any number of bits. The data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags.

ZERO BIT INSERTION

The flag has a unique binary bit pattern: 7E HEX. To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's.The 8274 performs zero bit insertion and deletion automatically in the SDLC/HDLC mode. The zero-bit stuffing ensures periodic transitions in the data stream. These transitions are necessary for a phase lock circuit, which may be used at the receiver end to generate a receive clock which is in phase to the received data. The inserted and deleted 0's are not included in the CRC checking. The *address* field is used to address a given secondary station. The *control* field contains the link-level control information which includes implied acknowledgement, supervisory commands and responses, etc. A more detailed discussion of higher level protocol functions is beyond the scope of this application note. Interested readers may refer to the references at the end of this application note.

The *data field* may be of any length and content in HDLC. Note that SDLC specifies that data field be a multiple of bytes only. In data communications, it is gen-

erally desirable to transmit data which may be of any content. This requires that data field should not contain characters which are defined to assist the transmission protocol (like opening flag 7EH in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion discussed earlier and the bit orientated nature of the protocol.

The last field is the FCS (Frame Check Sequence). The FCS uses the error detecting techniques called Cyclic Redundancy Check. In SDLC/HDLC, the CCITT-CRC must be used.

NON-RETURN TO ZERO INVERTED (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC protocol. It allows HDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for a phase lock circuit at the receiver end to derive a receive clock (from received data) which is synchronized to the received data and at the same time ensure data transparency.

Byte Synchronous Communication

As the name implies, Byte Synchronous Communication is a synchronous communication protocol which means that the transmitting station is synchronized to the receiving station through the recognition of a special sync character or characters. Two examples of Byte Synchronous protocol are the IBM Bisync and Monosync. Bisync has two starting sync characters per message while monosync has only one sync character. For the sake of abrevity, we will only discuss Bisync here. All the discussion is valid for Monosync also. Any exceptions will be noted. Figure 2 shows a typical Bisync message format.

The Bisync protocol is defined for half duplex communication between two or more stations over point to point or multipoint communication lines. Special characters control link access, transmission of data and termination of transmission operations for the system. A detailed discussion of these special control characters (SYN, ENQ, STX, ITB, ETB, ETX, DLE, SOH, ACK0, ACK1, WACK, NAK and EOT, etc) is beyond the scope of this Application Note. Readers interested in more detailed discussion are directed to the references listed at the end of this Application Note.

As shown in Figure 2, each message is preceded by two sync characters. Since the sync characters are defined at the beginning of the message only, the transmitter must insert fill characters (sync) in order to maintain synchronization with the receiver when no data is being transmitted.

TRANSPARENT TRANSMISSION

Bisync protocol requires special control characters to maintain the communication link over the line. If the data is EBCDIC encoded, then transparency is ensured by the fact that the data field will not contain any of the bisync control characters. However, if data does not conform to standard character encoding techniques, transparency in bisync is achieved by inserting a special character DLE (Data Link Escape) before and after a string of characters which are to be transmitted transparently. This ensures that any data charaters which match any of the special characters are not confused for special characters. An example of a transparent block is shown in Figure 3.

In a transparent mode, it is required that the CRC(BCC) is not performed on special characters. Later on, we will show how the 8274 can be used to achieve transparent transmission in Bisync mode.

SYNC	SYNC	SOH	HEADER	STX TEXT	ETX OR ETB	CRC 1	CRC 2

Figure 2. Bisync Message Format

DLE	STX	TRANSPARENT TRANSMISSION	DLE	ETX	BCC

Enter transparent mode　　　　return to normal mode

Figure 3. Bisync Transparent Format

BLOCK DIAGRAM

This section discusses the block diagram view of the 8274. The CPU interface and serial interface is discussed separately. This will be followed by a hardware example in the fifth section, which will show how to interface the 8274 with the Intel CPU 8088. The 8274 block diagram is shown in Figure 4.

CPU Interface

The CPU interface to the system interface logic block utilizes the A0, A1, $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$ inputs to communicate with the internal registers of the 8274. Figure 5 shows the address of the internal registers. The DMA interface is achieved by utilizing DMA request lines for each channel: $TxDRQ_A$, $TxDRQ_B$, $RxDRQ_A$, $RxDRQ_B$. Note that

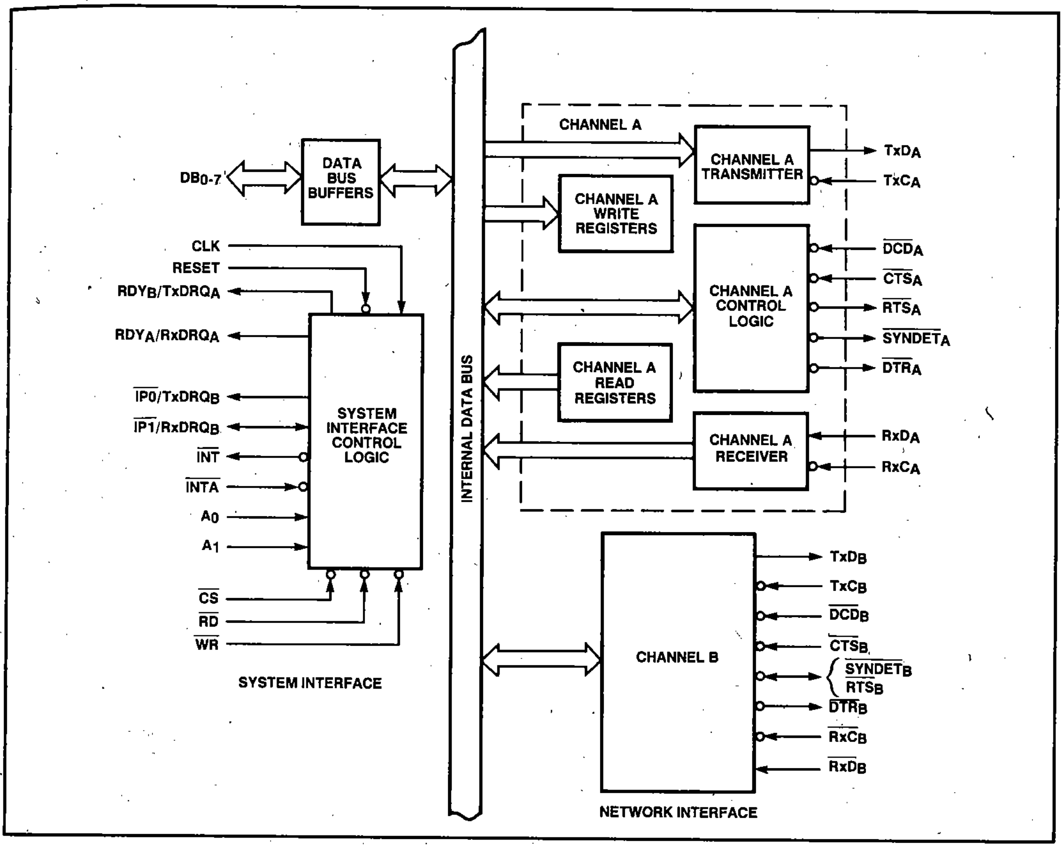


Figure 4. 8274 Block Diagram

$\overline{CS}$	A1	A0	Read Operation	Write Operation
0 0	0 1	0 0	CHA DATA READ CHA STATUS REGISTER (RR0,RR1)	CHA DATA WRITE CHA COMMAND/PARAMETER (WR0–WR7)
0 0	0 1	1 1	CHB DATA READ CHB STATUS REGISTER (RR0,RR1,RR2)	CHB DATA WRITE CHB COMMAND/PARAMETER (WR0–WR7)
1	X	X	HIGH Z	HIGH Z

Figure 5. Bus Interface

$TxDRQ_B$ and $RxDRQ_B$ becomes IPO and IPI respectively in non-DMA mode. IPI is the Interrupt Priority Input and IPO is the Interrrupt Priority Output. These two pins can be used for connecting multiple MPSCs in a daisy chain. If the Wait Mode is programmed, then $TxRDQ_A$ and $RxDRQ_A$ pins become RDY_B and RDY_A pins. These pins can be wire-or'ed and are usually hooked up to the CPU RDY line to synchronize the CPU for block transfers. The INT pin is activated whenever the MPSC requires CPU attention. The INTA may be used to utilize the powerful vectored mode feature of the 8274. Detailed discussion on these subjects will be done later in this Application Note. The Reset pin may be used for hardware reset while the clock is required to click the internal logic on the MPSC.

Serial Interface

On the serial side, there are two completely independent channels: Channel A and Channel B. Each channel consists of a transmitter block, receiver block and a set of read/write registers which are used to initialize the device. In addition, a control logic block provides the modem interface pins. Channel B serial interface logic is a mirror image of Channel A serial interface logic, except for one exception: there is only one pin for RTS_B and $SYNDET_B$. At a given time, this pin is either RTS_B or $SYNDET_B$. This mode is programmable through one of the internal registers on the MPSC.

Transmit And Receive Data Path

Figure 6 shows a block diagram for transmit and receive data path. Without describing each block on the diagram, a brief discussion of the block diagram will be presented here.

TRANSMIT DATA PATH

The transmit data is transferred to the twenty-bit serial shift register. The twenty-bits are needed to store two bytes of sync characters in bisync mode. The last three bits of the shift register are used to indicate to the internal control logic that the current data byte has been shifted out of the shift register. The transmit data in the transmit shift register is shifted out through a two bit delay onto the TxData line. This two bit delay is used to synchronize the internal shift clock with the external transmit clock. The data in the shift register is also presented to zero bit insertion logic which inserts a zero after sensing five contiguous ones in the data stream. In parallel to all this activity, the CRC-generator is computing CRC on the transmitted data and appends the frame with CRC bytes at the end of the data transmission.

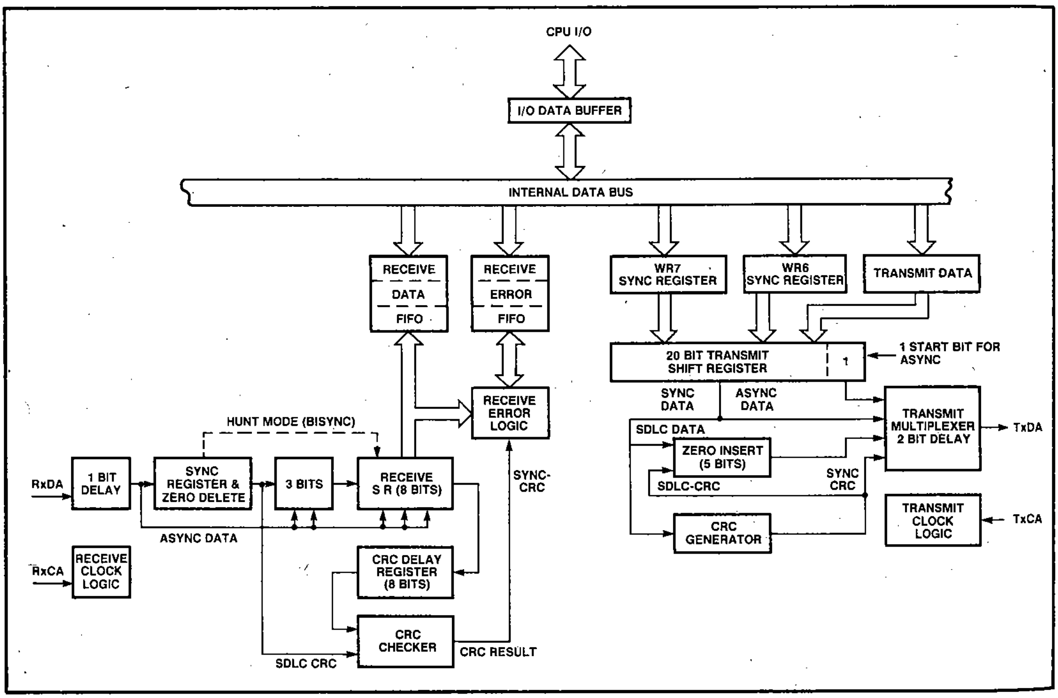


Figure 6. Transmit and Receive Data Path

RECEIVE DATA PATH

The received data is passed through a one-bit delay before it is presented for flag/sync comparison. In bisync mode, after the synchronization is achieved, the incoming data bypasses the sync register and enters directly into the three bit buffer on its way to receive shift register. In SDLC mode, the incoming data always passes through the sync register where data pattern is continously monitored for contiguous ones for zero deletion logic. The data then enters the three bit buffer and the receive shift register. From the receive shift register, the data is transferred to the three byte deep FIFO. The data is transferred to the top of the FIFO at the chip clock rate (not the receiver clock). It takes three chip clock/periods to transfer data from the serial shift register to the top of the FIFO. The three bit deep Receive Error FIFO shifts any error condition which may have occured during a frame reception. While all this is happening, the CRC checker is checking the CRC on the incoming data. The computed CRC is checked with the CRC bytes attached to the incoming frame and an error generated under a no-check condition. Note that the bisync data is presented to the CRC checker with an 8-bit delay. This is necessary to achieve transparency in bisync mode as will be shown later in this Application Note.

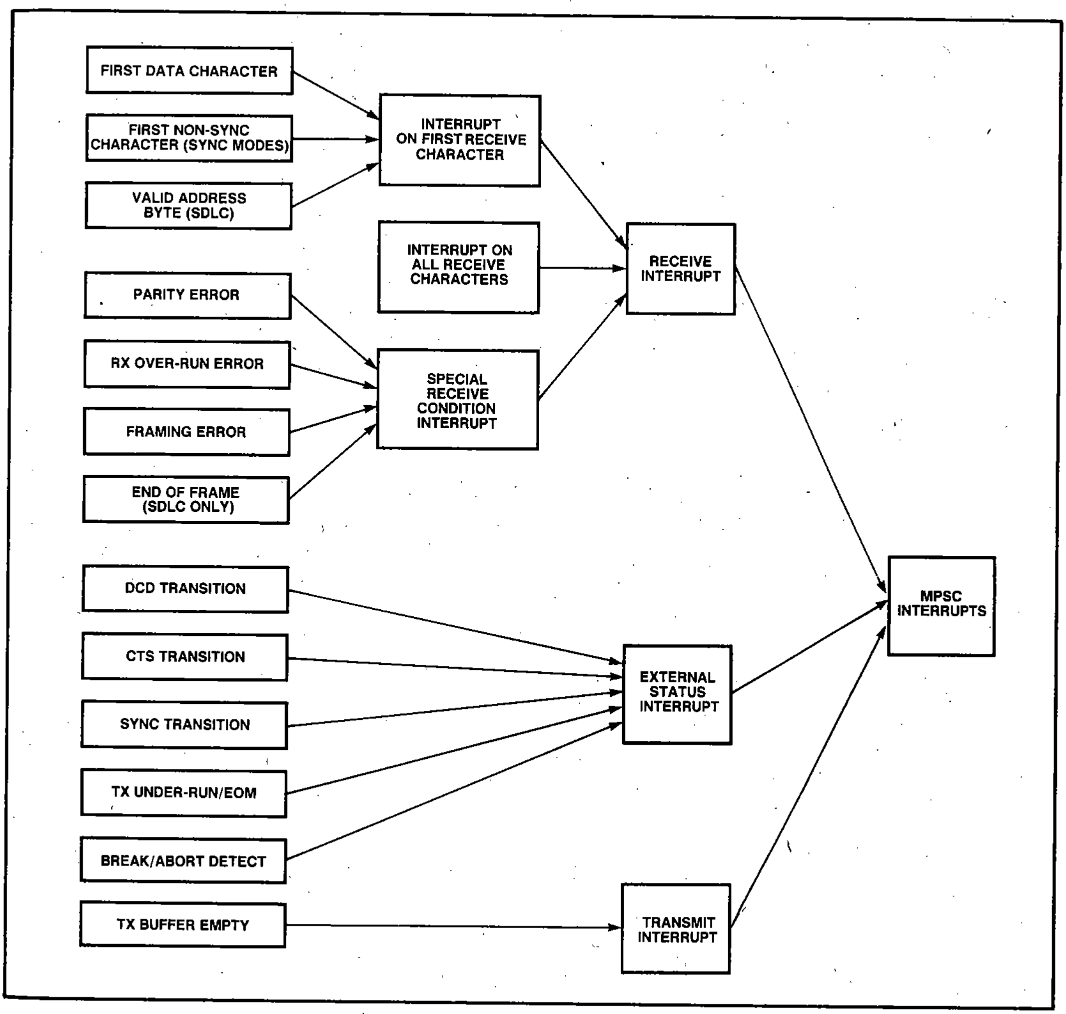


Figure 7. MPSC Interrupt Structure

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) INTERRUPT STRUCTURE

The MPSC offers a very powerful interrupt structure, which helps in responding to an interrupt condition very quickly. There are multiple sources of interrupts within the MPSC. However, the MPSC resolves the priority between various interrupting sources and interrupts the CPU for service through the interrupt line. This section presents a comprehensive discussion on all the 8274 interrupts and the priority resolution between these interrupts.

All the sources of interrupts on the 8274 can be grouped into three distinct catagories. (See Figure 7)

1. Receive Interrupts
2. Transmit Interrupts
3. External/Status Interrupts.

An internal interrupt priority structure sets the priority between the interrupts. There are two programmable options available on the MPSC. The priority is set by WR2A, D2. (Figure 8)

PRIORITY

WR2A:D2	Highest					Lowest
0	RxA	TxA	RxB	TxB	EXTA	EXTB
1	RxA	TxA	RxB	TxB	EXTA	EXTB

Figure 8. Interrupt Priority

Receive Interrupt

All receive interrupts may be catagorized into two distinct groups: Receive Interrupt on Receive Character and Special Receive Condition Interrupts.

RECEIVE INTERRUPT ON RECEIVE CHARACTER

A receive interrupt is generated when a character is received by the MPSC. However, as will be discussed later, this is a programmable feature on the MPSC . A Rx character available interrupt is generated by the MPSC after the receive character has been assembled by the MPSC. It may be noted that in DMA transfer mode too, a receive interrupt on the first receive character should be programmed. In SDLC mode, if address search mode has been programmed, this interrupt will be generated only after a valid address match has occured. In bisync mode, this interrupt is generated on receipt of a character after at least two valid sync characters. In monosync mode, a character followed by at least a single valid sync character will generate this interrupt. An interrupt on first receive character signifies the beginning of a valid frame. An end of the frame is characterized by an "End of Frame" Interrupt (RR1: D7).* This bit (RR1:D7) is set in SDLC/HDLC mode only and signifies that a valid ending flag (7EH) has been received. This bit gets reset either by an "Error Reset" command (WR0: D5D4D3 = 110) or upon reception of the first character of the next frame. In multiframe reception, on receiving the interrupt at the "End of Frame" the CPU may issue an Error Reset command which will reset the interrupt. In DMA mode, the interrupt on first receive character is accompanied by a RxDRQ (Receiver DMA request) on the appropriate channel. At the end of the frame, an End of Frame interrupt is generated. The CPU may use this interrupt to jump into a routine which may redefine the receive buffer for the next incoming frame.

*RR1:D7 is bit D7 in Read Register 1.

SPECIAL RECEIVE CONDITION INTERRUPTS

So far, we have assumed that the reception is error free. But this is not a 'typical' case in most real life applications. Any error condition during a frame reception generates yet another interrupt — special receive condition interrupt. There are four different error conditions which can generate this interrupt.

(i) Parity error
(ii) Receive Overrun error
(iii) Framing error
(iv) End of Frame

(i) Parity error: Parity error is encountered in asynchronous (start-stop bits) and in bisync/monosync protocols. Both odd or even parity can be programmed. A parity error in a received byte will generate a special receive condition interrupt and sets bit 4 in RR1.

(ii) Receive Overrun error: If the CPU or the DMA controller (in DMA mode) fails to read a received character within three byte times after the received character interrupt (or DMA request) was generated, the receiver buffer will overflow and this will generate a special receive condition interrupt and sets bit 5 in RR1.

(iii) Framing error: In asynchronous mode, a framing error will generate a spécial receive interrupt and set bit D6 in RR1. This bit is not latched and is updated on the next received character.

(iv) End of frame: This interrupt is encountered in SDLC/HDLC mode only. When the MPSC receives the closing flag, it generates the special receive condition interrupt and sets bit D7 in RR1.

All the special receive condition interrupts may be reset by issuing an Error Reset Command.

CRC Error: In SDLC/HDLC and synchronous modes, a CRC error is indicated by bit D6 in RR1. When used to check CRC error, this bit is normally set until a correct CRC match is obtained which resets this bit. After receiving a frame, the CPU must read this bit (RR1:D6) to determine if a valid CRC check had occured. It may be noted that a CRC error does not generate an interrupt.

It may be also be pointed out that in SDLC/HDLC mode, receive DMA requests are disabled by a special receive condition and can only be re-enabled by issuing an Error Reset Command.

Transmit Interrupt

A transmit buffer empty generates a transmit interrupt. This has been discussed earlier under "Transmit in Interrupt Mode" and it would be sufficient to note here that a transmit buffer empty interrupt is generated only when the transmit buffer gets empty — assuming it had a data character loaded into it earlier. This is why on starting a frame transmission, the first data character is loaded by the CPU without a transmit empty interrupt (or DMA request in DMA mode). After this character is loaded into the serial shift register, the buffer becomes empty, and an interrupt (or DMA request) is generated. This interrupt is reset by a "Reset Tx Interrupt/DMA Pending" command (WR0: D5 D4 D3 = 101).

External/Status Interrupt

Continuing our discussion on transmit interrupt, if the transmit buffer is empty and the transmit serial shift register also becomes empty (due to the data character shifted out of the MPSC), a transmit under-run interrupt will be generated. This interrupt may be reset by "Reset/External Status Interrupt" command (WR0: D5 D4 D3 = 101).

The External Status Interrupt can be caused by five different conditions:
(i) DCD Transition
(ii) CTS Transition
(iii) Sync/Hunt Transition
(iv) Tx under-run/EOM condition
(v) Break/Abort Detection.

DCD,CTS TRANSITION

Any transition on these inputs on the serial interface will generate an External/Status interrupt and set the corresponding bits in status register RR0. This interrupt will also be generated in DMA as well as in Wait Mode. In order to find out the state of the CTS or DCD pins *before* the transition had occurred, RR0 must be read before issuing a Reset External/Status Command through WR0. A read of RR0 after the Reset External/Status Command will give the condition of CTS or DCD pins *after* the transition had occurred. Note that bit D5 in RR0 gives the complement of the state of CTS pin while D3 in RR0 reflects the actual state of the DCD pin.

SYNC HUNT TRANSITION

Any transition on the SYNDET input generates an interrupt. However, sync input has different functions in different modes and we shall discuss them individually.

SDLC Mode

In SDLC mode, the SYNDET pin is an output. Status register RR1, D4 contains the state of the SYNDET pin. The Enter Hunt Mode initially sets this bit in R0. An opening flag in a received SDLC frame resets this bit and generates an external status interrupt. Every time the receiver is enabled or the Enter Hunt Code Command is issued, an external status interrupt will be generated on receiving a valid flag followed by a valid address/data character. This interrupt may be reset by the "Reset External Status Interrupt" command.

External SYNC Mode

The MPSC can be programmed into External Sync Mode by setting WR4, D5 D4 = 11. The SYNDET pin is an input in this case and must be held high until an external character synchronization is established. However, the External Sync mode is enabled by the Enter Hunt Mode control bit (WR3: D4). A high at the SYNDET pin holds the sync/Hunt bit (RR0,D4) in the reset state. When external synchronization is established, SYNDET must be driven low on second rising edge of RxC after the rising edge of RxC on which the last bit of sync character was received. This high to low transition sets the Sync/Hunt bit and generates an external status interrupt, which must be reset by the Reset External/Status command. If the SYNDET input goes high again, another External Status Interrupt is generated, which may be cleared by Reset External Status command.

Mono-Sync/Bisync Mode

SYNDET pin acts as an output in this case. The Enter Hunt Mode sets the Sync/Hunt bit in R0. Sync/Hunt bit is reset when the MPSC achieves character sysnchronization. This high to low transition will generate an external status interrupt. The SYNDET pin goes active every time a sync pattern is detected in the data stream. Once again, the external status interrupt may be reset by the Reset External Status command.

Tx UNDER-RUN/END OF MESSAGE (EOM)

The transmitter logic includes a transmit buffer and a transmit serial shift register. The CPU loads the character into the transmit buffer which is transferred into the transmit shift register to be shifted out of the MPSC. If the transmit buffer gets empty, a transmit buffer empty interrupt is generated (as discussed earlier). However, if the transmit buffer gets empty *and* the serial shift register gets empty, a transmit under-run condition will be created. This generates an External Status Interrupt and the interrupt can be cleared by the Reset External Status command. The status register RR0, D6 bit is set when the transmitter under-runs. This bit plays an important role in controlling a transmit operation, as will be discussed later in this application note.

BREAK/ABORT DETECTION

In asynchronous mode, bit D7 in RR0 is set when a break condition is detected on the receive data line. This also generates an External/Status interrupt which may be reset by issuing a Reset External/Status Interrupt command to the MPSC. Bit D7 in RR0 is reset when the break condition is terminated on the receive data line and this causes another External/Status interrupt to be generated. Again, a Reset External/Status Interrupt command will reset this interrupt and will enable the break detection logic to look for the next break sequence.

In SDLC Receive Mode, an Abort sequence (seven or more 1's) detection on the receive data line will generate an External/Status interrupt and set RR0,D7. A Reset External/Status command will clear this interrupt. However, a termination of the Abort sequence will generate another interrupt and set RR0, D7 again. Once again, it may be cleared by issuing Reset External/Status Command.

This concludes our discussion on External Status Interrupts.

Interrupt Priority Resolution

The internal interrupt priority between various interrupt sources is resolved by an internal prority logic circuit, according to the priority set in WR2A. We will now discuss the interrupt timings during the priority resolution. Figures 9 and 10 show the timing diagrams for vectored and non-vectored modes.

VECTORED MODE

We shall assume that the MPSC accepted an internal request for an interrupt by activating the internal INT signal. This leads to generating an external interrupt signal on the INT pin. The CPU responds with an interrupt acknowledge (INTA) sequence. The leading edge of the first INTA pulse sets an internal interrupt acknowledge signal (we will call it Internal INTA). Internal INTA is reset by the high going edge of the third INTA pulse. The MPSC will not accept any internal requests for an interrupt during the period when Internal INTA is active (high). The MPSC resolves the priority during various existing internal interrupt requests during the Interrupt Request Priority Resolve Time, which is defined as the time between the leading edge of the first INTA and the leading edge of the second INTA from the CPU. Once the internal priorities have been resolved, an internal Interrupt-in-service Latch is set. The external INT is also deactivated when the Interrupt-in-Service Latch is set.

The lower priority interrupt requests are not accepted internally until an EOI (WR0: D5 D4 D3 = 111) command is issued by the CPU. The EOI command enables the lower priority interrupts. However, a higher priority interrupt

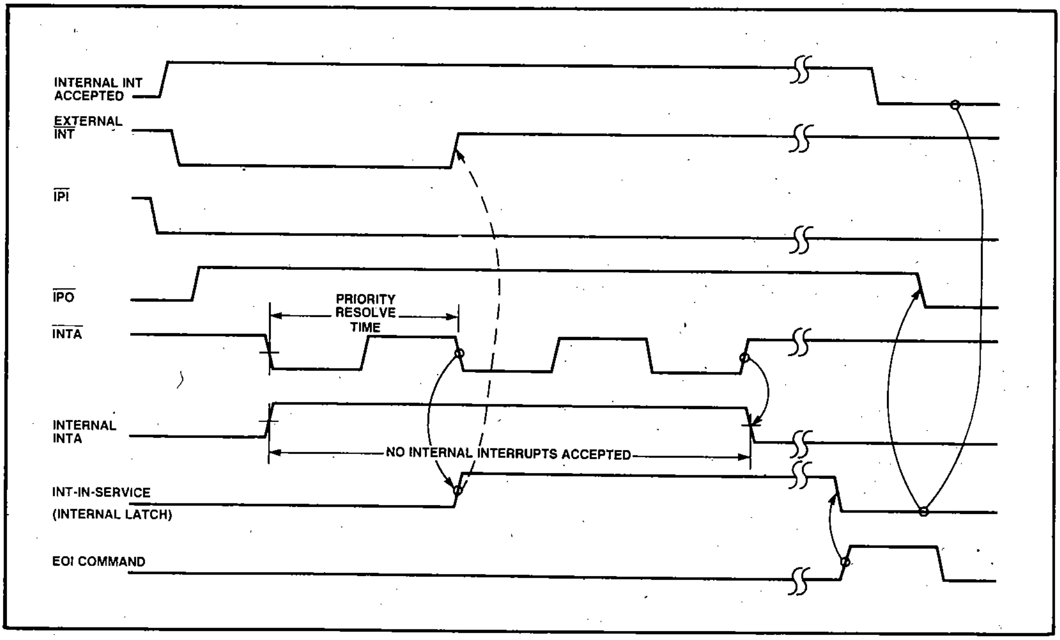


Figure 9. 8274 in 8085 Vectored Mode Priority Resolution Time

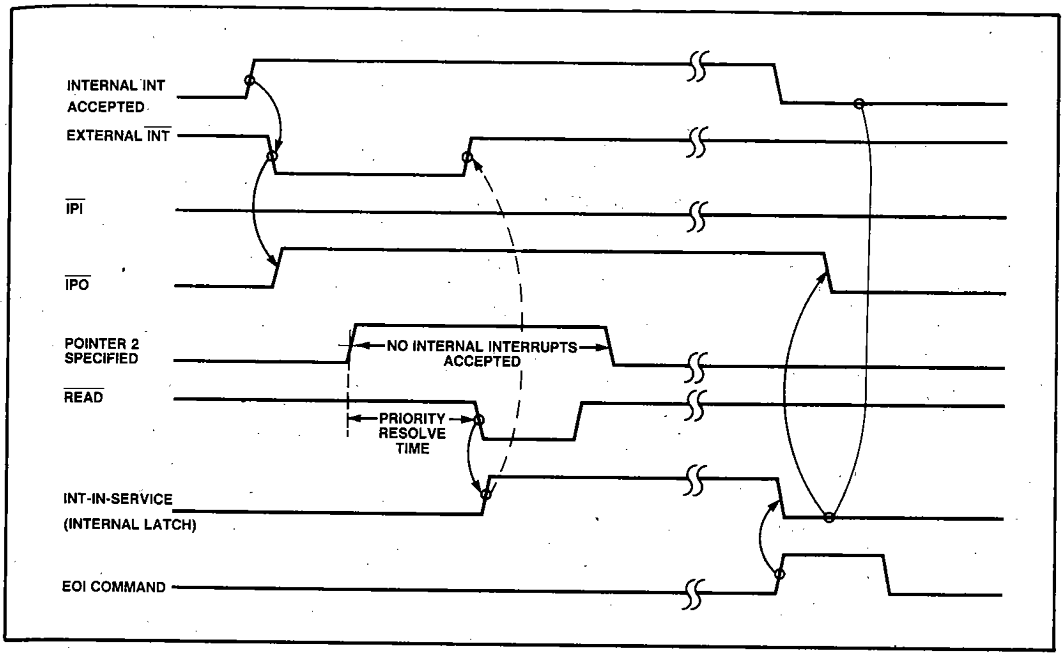


Figure 10. 8274 Non Vectored Mode Priority Resolve Time

request will still be accepted (except during the period when internal INTA is active) even though the Internal-in-Service Latch is set. This higher priority request will generate another external INT and will have to be handled by the CPU according to how the CPU is set up. If the CPU is set up to respond to this interrupt, a new INTA cycle will be repeated as discussed earlier. It may also be noted that a transmitter buffer empty and receive character available interrupts are cleared by loading a character into the MPSC and by reading the character received by the MPSC respectively.

NON-VECTORED MODE

Figure 10 shows the timing of interrupt sequence in non-vectored mode. The explanation for non-vectored is similar to the vector mode, except for the following exceptions.

— No internal priority requests are accepted during the time when pointer 2 for Channel B is specified.

— The interrupt request priority resolution time is the time between the leading edge of pointer 2 and leading edge of RD active. It may be pointed out that in non-vectored mode, it is assumed that the status affects vector mode is used to expedite interrupt response.

On getting an interrupt in non-vectored mode, the CPU must read status register RR2 to find out the cause of the interrupt. In order to do so, first a pointer to status register RR2 is specified and then the status read from RR2. It may be noted here that after specifying the pointer, the CPU must read status register RR2 otherwise, no new interrupt requests will be accepted internally.

Just like the vectored mode, no lower internal priority requests are accepted until an EOI command is issued by the CPU. A higher priority request can still interrupt the CPU (except during the priority request inhibit time). It is important to note here that if the CPU does not perform a read operation after specifying the pointer 2 for Channel B, the interrupt request accepted before the pointer 2 was activated will remain valid and no other request (high or low priority) will be accepted internally. In order to complete a correct priority resolution, it is advised that a read operation be done after specifying the pointer 2B.

IPI and IPO

So far, we have ignored the IPI and IPO signals shown in Figures 9 and 10. We may recall that IPI is the Interrupt-Priority-Input to the MPSC. In conjunction with the IPO (Interrupt Priority Output), it is used to daisy chain multiple MPSC's. MPSC daisy chaining will be discussed in detail later in this application note.

EOI Command

The EOI command as explained earlier, enables the lower priority interrupts by resetting the internal In-Service-Latch, which consequently resets the IPO output to a low state. See Figures 9 and 10 for details. Note that before issuing any EOI command, the internal interrupting source must be satisfied otherwise, same source will interrupt again. The Internal Interrupt is the signal which gets reset when the internal interrupting source is satisfied (see Figure 9).

This concludes our discussion on the MPSC Interrupt Structure.

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) MODES OF OPERATION

The MPSC provides two fully independent channels that may be configured in various modes of operations. Each channel can be configured into full duplex mode and may operate in a mode or protocol different from the other channel. This feature will be very efficient in an application which requires two data link channels operating in different protocols and possibly at different data rates. This section presents a detailed discussion on all the 8274 modes and shows how to configure it into these modes.

Interrupt Driven Mode

In the interrupt mode, all the transmitter and receiver operations are reported to the processor through interrupts. Interrupts are generated by the MPSC whenever it requires service. In the following discusson, we will discuss how to transmit and receive in interrupt driven mode.

TRANSMIT IN INTERRUPT MODE

The MPSC can be configured into interrupt mode by appropriately setting the bits in WR2 A (Write Register 2, Channel A). Figure 11 shows the modes of operation.

WR2A		
D1	**D0**	**MODE**
0	0	CH A and CH B in Interrupt Mode
0	1	CH A in DMA and CH B in Interrupt Mode
1	0	CH A and CH B in DMA Mode
1	1	Illegal

Figure 11. MPSC Mode Selection for Channel A and Channel B.

We will limit our discussion to SDLC transmit and receive only. However, exceptions for other synchronous protocols will be pointed out. To initiate a frame transmission, the first data character must be loaded from the CPU, in all cases. (DMA Mode too, as you will notice later in this application note). Note that in SDLC mode, this first data character may be the address of the station addressed by the MPSC. The transmit buffer consists of a transmit buffer and a serial shift register. When the character is transferred from the buffer into the serial shift register, an interrupt due to transmit buffer empty is generated. The CPU has one byte time to service this interrupt and load another character into the transmitter buffer. The MPSC will generate an interrupt due to transmit buffer underrun condition if the CPU does not service the Transmit Buffer Empty Interrupt within one byte time.

This process will continue until the CPU is out of any more data characters to be sent. At this point, the CPU does not respond to the interrupt with a character but simply issues a Reset Tx INT/DMA pending command (WR0: D5 D4 D3 = 1 0 1). The MPSC will ultimately under-run, which simply means that both the transmit buffer and transmit shift registers are empty. At this point, flag character (7EH) or CRC byte is loaded into the transmit shift register. This sets the transmit under-run bit in RR0 and generates "Transmit Under-run/EOM" interrupt (RR0:D6=1).

You will recall that an SDLC frame has two CRC bytes after the data field. 8274 generates the CRC on all the data that is loaded from the CPU. During initialization, there is a choice of selecting a CRC-16 or CCITT-CRC (WR5: D2). In SDLC/HDLC operation, CCITT-CRC must be selected. We will now see how the CRC gets inserted at the end of the data field. Here we have a choice of having the CRC attached to the data field or sending the frame without the CRC bytes. During transmission, a "Reset Tx Under-run/EOM Latch" command (WR0: D7 D6 = 11) will ensure that at the end of the frame when the transmitter underruns, CRC bytes will be automatically inserted at the end of the data field. If the "Reset Tx Under-run/EOM Latch" command was not issued during the transmission of data characters, no CRC would be inserted and the MPSC will transmit flags (7EH) instead.

However, in case of CRC transmission, the CRC transmission sets the Tx Under-run/EOM bit and generates a Transmitter Under-run/EOM Interrupt as discussed earlier. This will have to be reset in the next frame to ensure CRC insertion in the next frame. It is recommended that Tx Under-run/EOM latch be reset very early in the transmission mode, preferably after loading the first character. It may be noted here that Tx Under-run/EOM latch cannot be reset, if there is no data in the transmit buffer. This means that at least one character has to be loaded into the MPSC before a "Reset Transmit Under-run/EOM Latch" command will be accepted by the MPSC.

When the transmitter is under-run, an interrupt is generated. This interrupt is generated at the beginning of the CRC transmission, thus giving the user enough time (minimum 22 transmit clock cycles) to issue an Abort

command (WR0: D5 D4 D3 = 0 0 1) in case if the transmitted data had an error. The Abort Command will ensure that the MPSC transmits at least eight 1's but less than fourteen 1's before the line reverts to continuous flags. The receiver will scratch this frame because of bad CRC.

However, assuming the transmission was good (no Abort Command issued), after the CRC bytes have been transmitted, closing flag (7EH) is loaded into the transmit buffer. When the flag (7EH) byte is transferred to the serial shift register, a transmit buffer empty interrupt is generated. If another frame has to be transmitted, a new data character has to be loaded into the transmit buffer and the complete transmit sequence repeated. If no more frames are to be transmitted, a "Reset Transmit INT/DMA Pending" command (WR0: D5 D4 D3 = 1 0 1) will reset the transmit buffer empty interrrupt.

For character oriented protocols (Bisync, Monosync), the same discussion is valid, except that during transmit under-run condition and transmit under-run/EOM bit in set state, instead of flags, filler sync characters are transmitted.

CRC Generation:

The transmit CRC enable bit (WR5: D0) must be set before loading any data into the MPSC. The CRC generator must be reset to all 1's at the beginning of each frame before CRC computation has begun. The CRC computation starts on the first data character loaded from the CPU and continues until the last data character. The CRC generated is inverted before it is sent on the Tx Data line.

Transmit Termination:

A successful transmission can be terminated by issuing a "Reset Transmit Interrupt/DMA Pending" command, as discussed earlier. However, the transmitter may be disabled any time during the transmission and the results will be as shown in Figure 12.

Transmitter Disabled during	Result
1. Data Transmission	Tx Data will send idle characters* which will be zero inserted.
2. CRC Transmission	16 bit transmission, corresponding to 16 bits of CRC will be completed. However, flag bits will be substituted in the CRC field.
3. Immediately after issuing ABORT command.	Abort will still be transmitted — output will be in the mark state.

Figure 12. Transmitter Disabled During Transmission

*Idle characters are defined as a string of 15 or more contiguous ones.

RECEIVE IN INTERRUPT MODE

The receiver has to be initialized into the appropriate receive mode (see sample program later in this application note). The receiver must be programmed into Hunt Mode (WR3: D4) before it is enabled (WR3: D0). The receiver will remain in the Hunt Mode until a flag (or sync character) is received. While in the SDLC/Bisync/Monosync mode, the receiver does not enter the Hunt Mode unless the Hunt bit (WR3, D4) is set again or the receiver is enabled again.

SDLC Address byte is stored in WR6. A global address (FFH) has been hardwired on the MPSC. In address search mode (WR3: D2 = 1), any frame with address matching with the address in WR6 will be received by the MPSC. Frames with global address (FFH) will also be received, irrespective of the condition of address search mode bit (WR3: D2). In general receive mode (WR3: D2=0), all frames will be received.

Since the MPSC only recognizes single byte address field, extended address recognition will have to be done by the CPU on the data passed on by the MPSC. If the first address byte is checked by the MPSC, and the CPU determines that the second address byte does not have the correct address field, it must set the Hunt Mode (WR3: D2 = 1) and the MPSC will start searching for a new address byte preceded by a flag.

Programmable Interrupts: The receiver may be programmed into any one of the four modes. See Figure 13 for details.

WR1, CHA		
D4	D3	Rx Interrupt Mode
0	0	Rx INT/DMA disable
0	1	Rx INT on first character
1	0	INT on all Rx characters (Parity affects vector)
1	1	INT on all Rx characters (Parity does not affect vector)

Figure 13. Receiver Interrupt Modes

All receiver interrupts can be disabled by WR1: D4 D3 = 0 0. Receiver interrupt on first character is normally used to start a DMA transfer or a block transfer sequence using WAIT to synchronize the data transfer to received or transmitted data.

External Status Interrupts:

Any change in DCD input or Abort detection in the received data, will generate an interrupt if External Status Interrupt was enabled (WR1: D0).

Special Receive Conditions:

The receiver buffer is quadruply buffered. If the CPU fails to respond to "receive character" available interrupt within a period of three byte times (received bytes), the receiver buffer will overflow and generate an interrupt. Finally, at the end of the received frame, an interrupt will be generated when a valid ending flag has been detected.

Receive Character Length:

The receive character length (6,7 or 8 bits/character) may be changed during reception. However, to ensure that the change is effective on the next received character, this must be done fast enough such that the bits specified for the next character have not been assembled.

CRC Checking:

The opening flag in the frame resets the receive CRC generator and any field between the opening and closing flag is checked for the CRC. In case of a CRC error, the CRC/Framing Error bit in status register 1 is set (RR1: D6=1). Receiver CRC may be disabled/enabled by WR3,D3. The CRC bytes on the received frame are passed on to the CPU just like data, and may be discarded by the CPU.

Receive Terminator:

An end of frame is indicated by End of Frame interrupt. The CPU may issue an "Error Reset" command to reset this interrupt.

DMA (Direct Memory Access) Mode

The 8274 can be interfaced directly to the Intel DMA Controllers 8237A, 8257A and Intel I/O Processor 8089. The 8274 can be programmed into DMA mode by setting appropriate bits in WR2A. See Figure 11 for details.

TRANSMIT IN DMA MODE:

After initializing the 8274 into the DMA mode, the first character must be loaded from the CPU to start the DMA cycle. When the first data character (may be the address byte in SDLC) is tranferred from the transmit buffer to the transmit serial shift register, the transmit buffer gets empty and a transmit DMA request (TxDRQ) is generated for the channel. Just like the interrupt mode, to ensure that the CRC bytes are included in the frame, the transmit under-run/EOM latch must be reset. This should preferably be done after loading the first character from the CPU. The DMA will progress without any CPU intervention. When the DMA controller reaches the terminal count, it will not respond to the DMA request, thus letting the MPSC under-run. This will ensure CRC transmission. However, the under-run condition will generate an interrupt due to the Tx under-run/EOM bit getting set (RR0: D6). The CPU should issue a "Reset TxInt/DRQ pending" command to reset TxDRQ and issue a "Reset External Status" command to reset Tx Under-run/EOM interrupt. Following the CRC transmission, flag (7EH) will be loaded into the transmit buffer. This will also generate the TxDRQ since the transmit buffer is empty following the transmission of the CRC bytes. The CPU may issue a "Reset TxINT/DRQ pending" command to reset the TxDRQ. "Reset TxINT/DRQ pending" command must be issued before setting up the transmit DMA channel on the DMA Controller, otherwise the MPSC will start the DMA transfer immediately after the DMA channel is set up.

RECEIVE IN DMA MODE

The receiver must be programmed in RxINT on first receive character mode (WR1: D4 D3 = 0 1). Upon receiving the first character, which may be the address byte in SDLC, the MPSC generates an interrupt and also generates a Rx DMA Request (Rx DRQ) for the appropriate channel. The CPU has three byte times to service this interrupt (enable the DMA controller, etc.) before the receiver buffer will overflow. It is advisable to initialize the DMA controller before receiving the first character. In case of high bit rates, the CPU will have to service the interrupt very fast in order to avoid receiver over-run.

Once the DMA is enabled, the received data is transfered to the memory under DMA control. Any received error conditions or external status change condition will generate an interrupt as in the interrupt driven mode. The End of Frame is indicated by the End of Frame interrupt which is generated on reception of the closing flag of the SDLC frame. This End of Frame condition also disables the Receive DMA request. The End of Frame interrupt may be reset by issuing an "Error Reset" command to the MPSC. The "Error Reset" command also re-enables the Receive DMA request. It may be noted that the End of Frame condition sets bits D7 in RR1. This bit gets reset by "Error Reset" command. However, End of Frame bit (RR1:D7) can also be reset by the flag of the next incoming frame. For proper operation, Error Reset Command should be issued "after" the End of Frame Bit (RR1:D7) is set. In a more general case, "Error Reset" command should be issued after End of Frame, Receive over-run or Receive parity bit are set in RR1.

Wait Mode

The wait mode is normally used for block transfer by synchronizing the data transfer through the Ready output from the MPSC, which may be connected to the Ready input of the CPU. The mode can be programmed by WR 1, D7 D5 and may be programmed separately and independently on CH A and CH B. The Wait Mode will be operative if the following conditions are satisfied.

(i) Interrupts are enabled.
(ii) Wait Mode is enabled (WR1: D7)
(iii) CS = 0, AI = 0

The RDY output becomes active when the transmitter buffer is full or receiver buffer is empty. This way the RDY output from the MPSC can be used to extend the CPU read and write cycle by inserting WAIT states. RDY_A or RDY_B are in high impedance state when the corresponding channel is not selected. This makes it possible to connect RDY_A and RDY_B outputs in wired OR configuration. Caution must be exercised here in using the RDY outputs of the MPSC or else the CPU may hang up for indefinite period. For example, let us assume that transmitter buffer is full and RDY_A is active, forcing the CPU into a wait state. If the CTS goes inactive during this period, the RDY_A will remain active for indefinite period and CPU will continue to insert wait states.

Vectored/Non-Vectored Mode

The MPSC is capable of providing an interrupt vector in response to the interrupt acknowledge sequence from the CPU. WR2, CH B contains this vector and the vector can be read in status register RR2. WR2, CH A (bit D5) can program the MPSC in vectored or non-vectored mode. See Figure 14 for details.

In both cases, WR2 may still have the vector stored in it. However, in vectored mode, the MPSC will put the vector on the data bus in response the INTA (Interrupt Acknowledge) sequence as shown in Figure 15. In non-vectored mode, the MPSC will not respond to the INTA sequence. However, the CPU can read the vector by polling Status Register RR2. WR2A, D4 and D3 can be programmed to respond to 8085 or 8086 INTA sequence. It may be noted here that IPI (Interrupt Priority In) pin on the MPSC must be active for the vector to appear on the data bus.

WR2A,D5	Interrupt Mode
0	Non-vectored Interrupt
1	Vectored Interrupt

Figure 14. Vectored Interrupts

Status Affect Vector

The vector stored in WR2B can be modified by the source of the interrupt. This can be done by setting the Status Affect Vector bit (WR1: D2). This powerful feature of the MPSC provides fast interrupt response time, by eliminating the need of writing a routine to read the status of the MPSC. Three bits of the vector are modified in eight different ways as shown on Figure 16. Bits V4,V3,V2 are modified in 8085 based system and bits V2, V1, V0 are modified in 8086/88 based system.

In non-vectored mode, the status affect vector mode can still be used and the vector read by the CPU. Status Register RR2B (Read Register 2 in Channel B) will contain this modified vector.

WR2A D5	D4	D3	IPI	MODE	1ST INTA	2ND INTA	3RD INTA
0	X	X	X	NON-VECTORED	HIGH-Z	HI-Z	HI-Z
1	0	0	0	8085-1	1100 1101	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	0	1	8085-1	1100 1101	HI-Z	HI-Z
1	0	1	0	8085-2	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	1	1	8085-2	HI-Z	HI-Z	HI-Z
1	1	0	0	8086	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	—
1	1	0	1	8086	HI-Z	HI-Z	—

Figure 15. MPSC Vectored Interrupts

(8085) V4 / (8086) V2	V3 / V1	V2 / V0	Channel	Interrupt Source
0	0	0	B	Tx BUFFER EMPTY
0	0	1		EXT/STAT CHANGE
0	1	0		RX CHAR AVAILABLE
0	1	1		SPECIAL Rx CONDITION
1	0	0	A	Tx BUFFER EMPTY
1	0	1		EXT/STAT CHANGE
1	1	0		RX CHAR AVAILABLE
1	1	1		SPECIAL Rx CONDITION

Rx Special Condition: Parity Error, Framing Error, Rx Over-run Error, EOF (SDLC)

EXT/STAT Change: Change in Modem Control Pin Status: CTS, DCD, SYNC, EOM, Break/Abort Detection

Figure 16. Status Affect Vector Mode

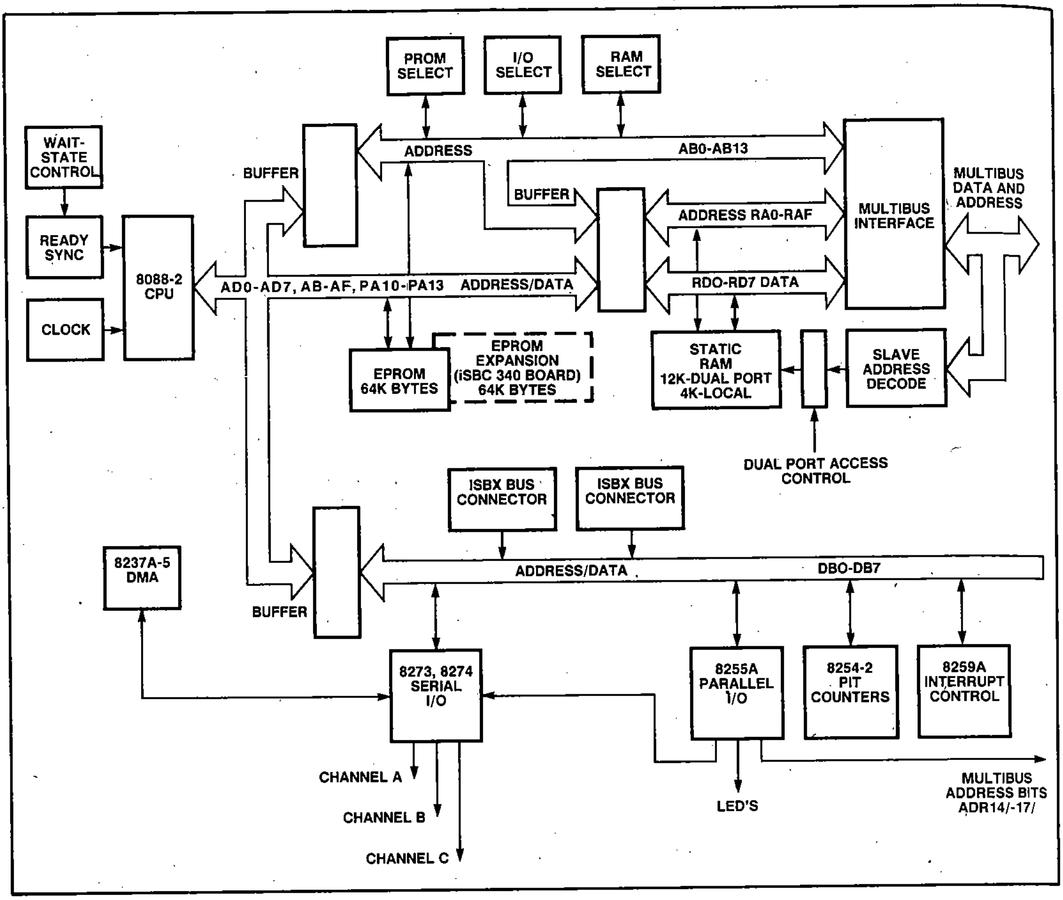


Figure 17. Functional Block Diagram — iSBC® 88/45

APPLICATION EXAMPLE

This section describes the hardware and software of an 8274/8088 system. The hardware vehicle used is the INTEL Single Board Computer iSBC 88/45 - Advanced Communication Controller. The software which exercises the 8274 is written in PLM 86.This example will demonstrate how 8274 can be configured into the SDLC mode and transfer data through DMA control. The hardware example will help the reader configure his hardware and the software examples will help in developing an application software. Most software examples closely approximates a real data link controller software in the SDLC communication and may be used with very little modification.

iSBC® 88/45

A brief description of the iSBC 88/45 board will be presented here. For more detailed information on the board and the schematics, refer to Hardware Manual for the iSBC 88/45, Advanced Communication Controller. iSBC 88/45 is an intelligent slave/multimaster communication board based on the 8088 processor, the 8274 and the 8273 SDLC/HDLC controller. Figure 17 shows the functional block diagram of the board. The iSBC 88/45 has the following features.

- 8 MHz processor
- 16K bytes of static RAM (12K dual port)
- Multimaster/Intelligent Slave Multibus Interface
- Nine Interrupt Levels 8259A
- Two serial channels through 8274
- One Serial channel through 8273
- S/W programmable baud rate generator
- Interfaces: RS 232, RS422/449, CCITT V.24
- 8237A DMA controller
- Baud Rate to 800k Baud

```
INITIALIZE_8274:PROCEDURE PUBLIC;

/************************************************************************/
/*                                                                      */
/*      INITIALIZE THE 8274 FOR SDLC MODE                               */
/*                                                                      */
/*      1. RESET CHANNEL                                                */
/*      2. EXTERNAL INTERRUPTS ENABLED                                  */
/*      3. NO WAIT                                                      */
/*      4. PIN 10 = RTS                                                 */
/*      5. NON-VECTORED INTERRUPT-8086 MODE                             */
/*      6. CHANNEL A DMA, CH B INT                                      */
/*      7. TX AND RX = 8 BITS/CHAR                                      */
/*      9. ADDRESS SEARCH MODE                                          */
/*      10. CD AND CTS AUTO ENABLE                                      */
/*      11. X1 CLOCK                                                    */
/*      12. NO PARITY                                                   */
/*      13. SDLC/HDLC MODE                                              */
/*      14. RTS AND DTR                                                 */
/*      15. CCITT - CRC                                                 */
/*      16. TRANSMITTER AND RECEIVER ENABLED                            */
/*      17. 7EH = FLAG                                                  */
/*                                                                      */
/************************************************************************/

DECLARE C BYTE;

/* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B   */
/* FORMAT IS: WRITE REGISTER, REGISTER DATA       */
/* INITIALIZE CHANNEL A ONLY                      */

DECLARE TABLE_74_A(*) BYTE DATA
        (00H,18H,         /* CHANNEL RESET */
         00H,80H,         /* RESET TX CRC */
         02H,11H,         /* PIN 10=RTSB, A DMA, B INT */
         04H,20H,         /* SDLC/HDLC MODE, NO PARITY */
         07H,07EH,        /* SDLC FLAG */
         01H,0BH,         /* RX DMA ENABLE */
         05H,0EBH,        /* DTR, RTS, 8 TX BITS, TX ENABLE, */
                          /* SDLC CRC, TX CRC ENABLE */
         06H,55H,         /* DEFAULT ADDRESS */
         03H,0D9H,        /* 8 RX BITS, AUTO ENABLES, HUNT MODE, */
                          /* RX CRC ENABLE */
         0FFH);           /* END OF INITIALIZATION TABLE */

DECLARE TABLE_74_B(*) BYTE DATA
        (02H,00H,         /* INTERRUPT VECTOR */
         01H,1CH,         /* STATUS AFFECTS VECTOR */
         0FFH);           /* END */

/* INITIALIZE THE 8274 */

C=0;
DO WHILE TABLE_74_B(C) <> 0FFH;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
END;

C=0;
DO WHILE TABLE_74_A(C) <> 0FFH;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
END;
RETURN;
END INITIALIZE_8274;
```

Figure 18. Typical MPSC SDLC Initialization Sequence

For this application, the CPU is run at 8 MHz. The board is configured to operate the 8274 in SDLC operation with the data transfer in DMA mode using the 8237A. 8274 is configured first in non-vectored mode in which case the INTEL Priority Interrupt Controller 8259A is used to resolve priority betweeen various interrupting sources on the board and subsequently interrupt the CPU. However, the vectored mode of the 8274 is also verified by disabling the 8259A and reading the vectors from the 8274. Software examples for each case will be shown later.

The application example is interrupt driven and uses DMA for all data transfers under 8237A control. The 8254 provides the transmit and receive clocks for the 8274. The 8274 was run at 400K baud with a local loopback (jumper wire) on Channel A data.The board was also run at 800K baud by modifying the software as will be discussed later in the Special Applications section. One detail to note is that the Rx Channel DMA request line from the 8274 has higher priority than the Tx Channel DMA request line. The 8274 master clock was 4.0 MHz. The on-board RAM is used to define transmit and receive data buffers. In this application, the data is read from memory location 800H through 810H and transferred to memory location 900H to 910H through the 8274 Serial Link. The operation is full duplex. 8274 modem control pins, $\overline{CTS}$ and $\overline{CD}$ have been tied low (active).

Software

The software consists of a monitor program and a program to exercise the 8274 in the SDLC mode. Appendix A contains the entire program listing. For the sake of clarity, each source module has been rewritten in a simple language and will be discussed here individually. Note that some labels in the actual listings in the Appendix will not match with the labels here. Also the listing in the Appendix sets up some flags to communicate with the monitor. Some of these flags are not explained in detail for the reason that they are not pertinent to this discussion. The monitor takes the command from a keyboard and executes this program, logging any error condition which might occur.

8274 Initialization

The MPSC is initialized in the SDLC mode for Channel A. Channel B is disabled. See Figure 18 for the initialization routine. Note that WR4 is initialized before setting up the transmitter and receive parameters. However, it may also be pointed out that other than WR4, all the other registers may be programmed in any order. Also SDLC-CRC has been programmed for correct operation. An incorrect CRC selection will result in incorrect operation. Also note that receive interrupt on first receive character has been programmed although Channel A is in the DMA mode.

Interrupt Routines

The 8274 interrupt routines will be discussed here. On an 8274 interrupt, program branches off to the "Main Interrupt Routine". In main interrupt routine, status register RR2 is read. RR2 contains the modified vector. The cause of the interrupt is determined by reading the modified bits of the vector. Note that the 8274 has been programmed in the non-vectored mode and status affects vector bit has been set. Depending on the value of the modified bits, the appropriate interrupt routine is called. See Figure 19 for the flow diagram and Figure 20 for the source code. Note that an End of Interrupt Command is issued after servicing the interrupt. This is necessary to enable the lower priority interrupts.

Figure 21 shows all the interrupt routines called by the Main Interrupt Routine. "Ignore - Interrupt" as the name implies, ignores any interrupts and sets the FAIL flag. This is done because this program is for Channel A only and we are ignoring any Channel B interrupts. The important thing to note is the Channel A Receiver Character available routine. This routine is called after receiving the first character in the SDLC frame. Since the transfer mode is DMA, we have a maximum of three character times to service this interrupt by enabling the DMA controller.

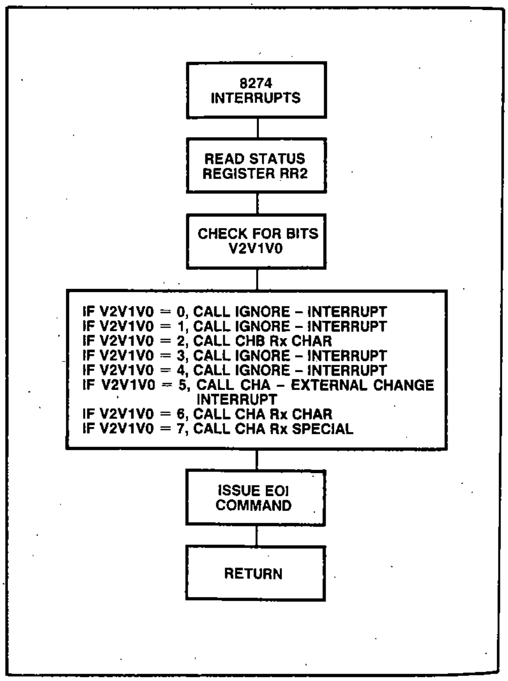


Figure 19. Interrupt Response Flow Diagram

```
/**************************/
/* MAIN INTERRUPT ROUTINE */
/**************************/

OUTPUT(COMMAND_B_74) = 2;                /* SET POINTER TO 2*/
TEMP = INPUT(STATUS_B_74) AND 07H;       /* READ INTERRUPT VECTOR */
                                         /* CHECK FOR CHA INT ONLY*/
/* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
DO CASE TEMP;
        CALL    IGNORE_INT;              /* V2V1V0 = 000*/
        CALL    IGNORE_INT;              /* V2V1V0 = 001*/
        CALL    CHB_RX_CHAR;             /* V2V1V0 = 010*/
        CALL    IGNORE_INT;              /* V2V1V0 = 011*/
        CALL    IGNORE_INT;              /* V2V1V0 = 100*/
        CALL    CHA_EXTERNAL_CHANGE;     /* V2V1V0 = 101*/
        CALL    CHA_RX_CHAR;             /* V2V1V0 = 110*/
        CALL    CHA_RX_SPECIAL;          /* V2V1V0 = 111*/
END;
OUTPUT(COMMAND_A_74) =38H;        /* END OF INTERRUPT FOR 8274 */
RETURN;
END INTERRUPT_8274;
```

Figure 20. Typical Main Interrupt Routine

```
/******************************************************/
/* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
/******************************************************/

CHA_EXTERNAL_CHANGE: PROCEDURE;

TEMP = INPUT(STATUS_A_74);        /* STATUS REG 1*/
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
        TXDONE_S=DONE;
ELSE DO;
        TXDONE_S=DONE;
        RESULTS_S=FAIL;
    END;
OUTPUT(COMMAND_A_74) = 10H;       /* RESET EXT/STATUS INTERRUPTS */
RETURN;
END CHA_EXTERNAL_CHANGE;
/**********************************************************/
/* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
/**********************************************************/

CHA_RX_SPECIAL: PROCEDURE;

        OUTPUT(COMMAND_A_74) = 1;
        TEMP = INPUT(STATUS_A_74);
        IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
              DO;
                IF(TEMP AND 040H) = 040H THEN
                  RESULTS_S = FAIL;         /* CRC ERROR */
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
              END;
              ELSE DO;
                IF (TEMP AND 20H) = 20H THEN DO;
                RESULTS_S = FAIL;           /* RX OVERRUN ERROR*/
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
                END;
              END;
RETURN;
END CHA_RX_SPECIAL;

/*****************************************/
/* CHANNEL A RECEIVE CHARACTER AVAILABLE */
/*****************************************/

CHA_RX_CHAR: PROCEDURE;
OUTPUT(SINGLE_MASK) = CHO_SEL;           /*ENABLE RX DMA CHANNEL*/
RETURN;
END CHA_RX_CHAR;
```

Figure 21. 8274 Typical Interrupt Handling Routines

It may be recalled that the receiver buffer is three bytes deep in addition to the receiver shift register. At very high data rates, it may not be possible to have enough time to read RR2, enable the DMA controller without overrunning the receiver. In a case like this, the DMA controller may be left enabled before receiving the Receive Character Interrupt. Remember, the Rx DMA request and interrupt for the receive character appears at the same time. If the DMA controller is enabled, it would service the DMA request by reading the received character. This will make the 8274 interrupt line go inactive. However, the 8259A has latched the interrupt and a regular interrupt acknowledge sequence still occurs after the DMA controller has completed the transfer and given up the bus. The 8259A will return Level 7 interrupt since the 8274 interrupt has gone away. The user software must take this into account, otherwise the CPU will hang up.

The procedure shown for the Special Receive Condition Interrupt checks if the interrupt is due to the End of Frame. If this is not TRUE, the FAIL flag is set and the program aborted. For a real life system, this must be followed up by error recovery procedures which obviously are beyond the scope of this Application Note.

The transmission is terminated when the End of Message (RR0, D6) interrupt is generated. This interrupt is serviced in the Channel A External/Status Change interrupt procedure. For any other change in external status conditions, the program is aborted and a FAIL flag set.

Main Program

Finally, we will briefly discuss the main program. Figure 22 shows the source program. It may be noted that the Transmit Under-run latch is reset after loading the first character into the 8274. This is done to ensure CRC transmission at the end of the frame. Also, the first character is loaded from the CPU to start DMA transfer of subsequent data. This concludes our discussion on hardware and software example. Appendix A also includes the software written to exercise the 8274 in the vectored mode by disabling the 8259A.

```
CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

        CALL     ENABLE_INTERRUPTS_S;
        CALL     INIT_8274_SDLC_S;
        ENABLE;
        OUTPUT(COMMAND_A_74) = 28H;    /* RESET TX INT/DMA          */
        OUTPUT(COMMAND_B_74) = 28H;    /* BEFORE INITIALIZING 8237*/
        CALL     INIT_8237_S;
        OUTPUT(DATA_A_74) = 55H;       /*LOAD FIRST CHARACTER FROM */
                                       /*CPU */
        /* TO ENSURE CRC TRANSMISSION, RESET TX UNDERRUN LATCH    */
        OUTPUT(COMMAND_A_74) = OCOH;
        RXDONE_S, TXDONE_S=NOT_DONE;   /* CLEAR ALL FLAGS          */
        RESULTS_S=PASS;                /* FLAG SET FOR MONITOR     */
        DO WHILE TXDONE_S=NOT_DONE;    /* DO UNTIL TERMINAL COUNT  */
        END;

        DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
        /* WAIT FOR CRC TO GET TRANSMITTED */
        /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
        END;
        DO WHILE RXDONE_S=NOT_DONE;    /* DO UNTIL TERMINAL COUNT  */
        END;
        CALL     STOP_8237_S;
END CHA_SDLC_TEST;
```

Figure 22. Typical 8274 Transmit/Receive Set-Up in SDLC Mode

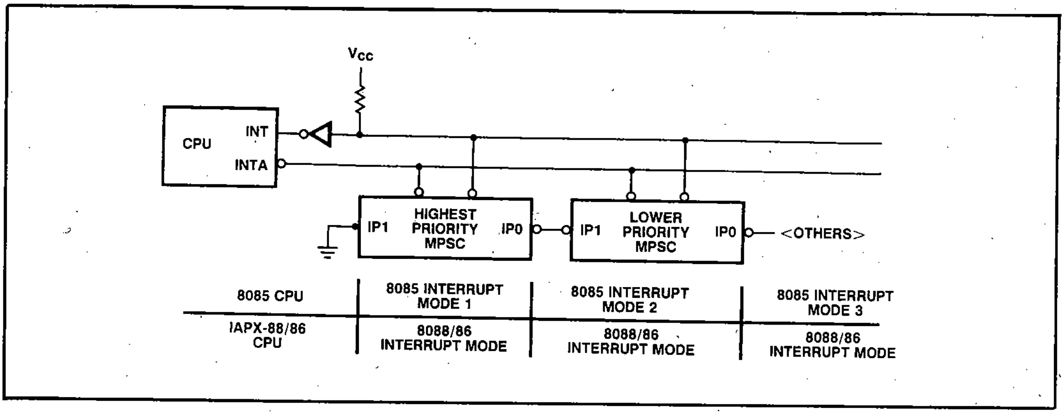


Figure 23. 8274 Daisy Chain Vectored Mode

SPECIAL APPLICATIONS

In this section, some special application issues will be discussed. This will be useful to a user who may be using a mode which is possible with the 8274 but not explicitly explained in the data sheet.

MPSC Daisy Chain Operation

Multiple MPSC can be connected in a daisy-chain configuration (see Figure 23). This feature may be useful in an application where multiple communication channels may be required and because of high data rates, conventional interrupt controller is not used to avoid long interrupt response times. To configure the MPSCs for the daisy chain operation, the interrupt priority input pins (IPI) and interrupt priority output pins (IPO) of the MPSC should be connected as shown. The highest priority device has its IPI pin connected to ground. Each MPSC is programmed in a vectored mode with status affects vector bit set. In the 8085 basic systems, only one MPSC should be programmed in the 8085 Mode 1. This is the MPSC which will put the call vector (CD Hex) on the data bus in response to the first INTA pulse (See Figure 15). It may be pointed out that the MPSC in 8085 Mode 1 will provide the call vector irrespective of the state of IPI pin. Once a higher priority MPSC generates an interrupt, its IPO pin goes inactive thus preventing lower priority MPSCs from interrupting the CPU. Preferably the highest priority MPSC should be programmed in 8085 Mode 1. It may be recalled that the Priority Resolve Time on a given MPSC extends from th falling edge of the first INTA pulse to the falling edge of the second INTA pulse. During this period, no new internal interrupt requests are accepted. The maximum number of the MPSCs that can be connected in a daisy chain is limited by the Priority Resolution Time. Figure 24 shows a maximum number of MPSCs that can be connected in various CPU systems. It may be pointed out that IPO to IPI delay time specification is 100ns.

Bisync Transparent Communication

Bisync applications generally require that data transparency be established during communication. This requires that the special control characters may not be included in the CRC accumulation. Refer to the Synchronous Protocol Overview section for a more detailed discussion on data transparency. The 8274 can be used for transparent communication in Bisync communications. This is made

System Configuration	Priority Resolution Time Min (ns)	Number of 8274s Daisy Chained (Max)
8086-1	400	4
8086-2	500	5
8086	800	8
8088	800	8
8085-2	1200	12
8085A	1920	19

Note: Zero wait states have been assumed.

Figure 24. 8274 Daisy Chain Operation

possible by the capability of the MPSC to selectively turn-on/turnoff the CRC accumulation while transmitting or receiving. In bisync transparent transmit mode, the special characters (DLE, DLE SYN etc) are excluded from CRC calculation. This can be easily accomplished by turning off the transmit CRC calculation (WR5: D5=0) before loading the special character into the transmit buffer. If the next character is to be included in the CRC accumulation, then the CRC can be enabled (WR5: D5=1). See Figure 25 for a typical flow diagram.

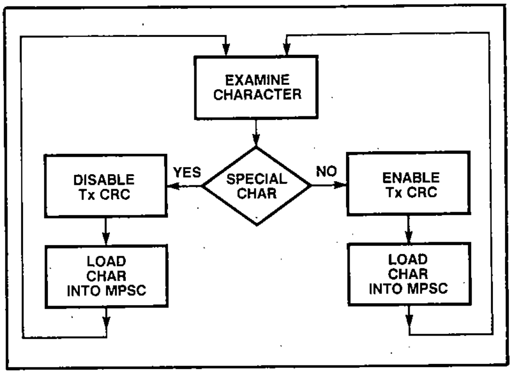


Figure 25. Transmit in Bisync Transparent Mode

During reception, it is possible to exclude received character from CRC calculation by turning off the Receive CRC after reading the special character. This is made possible by the fact that the received data is presented to receive CRC checker 8 bit times after the character has been received. During this 8 bit times, the CPU must read the character and decide if it wants it to be included in the CRC calculation. Figure 26 shows the typical flow diagram to achieve this.

It should be noted that the CRC generator must be enabled during CRC reception. Also, after reading the CRC bytes, two more characters (SYNC) must be read before checking for CRC check result in RR1.

Auto Enable Mode

In some data communication applications, it may be required to enable the transmitter or the receiver when the CTS or the DCD lines respectively, are activated by the modems. This may be done very easily by programming the 8274 into the Auto Enable Mode.The auto enable mode is set by writing a '1' to WR3,D5. The function of this mode is to enable the transmitter automatically when $\overline{CTS}$ goes active. The receiver is enabled when DCD goes active. An in-active state of $\overline{CTS}$ or $\overline{DCD}$ pin will disable the transmitter or the receiver respectively. However, the Transmit Enable bit (WR5:D3) and Receive Enable bit

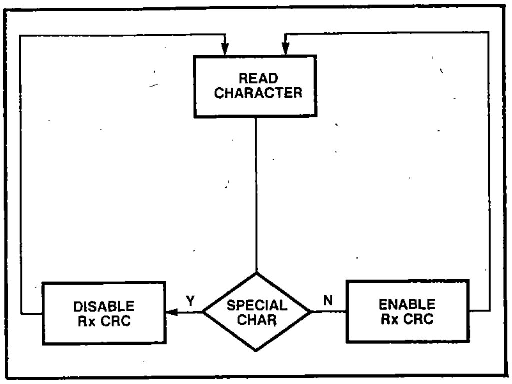


Figure 26. Receive in Bisync Transparent Mode

(WR3:D1) must be set in order to use the auto enable mode. In non-auto mode, the transmitter or receiver is enabled if the corresponding bits are set in WR5 and WR3, irrespective of the state $\overline{CTS}$ or $\overline{DCD}$ pins. It may be recalled that any transition on $\overline{CTS}$ or $\overline{DCD}$ pin will generate External/Status Interrupt with the corresponding bits set in RR1. This interrupt can be cleared by issuing a Reset External/Status interrupt command as discussed earlier.

Note that in auto enable mode, the character to be transmitted must be loaded into the transmit buffer after the $\overline{CTS}$ becomes active, not before. Any character loaded into the transmit buffer before the $\overline{CTS}$ became active will not be transmitted.

High Speed DMA Operation

In the section titled Application Example, the MPSC has been programmed to operate in DMA mode and receiver is programmed to generate an interrupt on the first receive character. You may recall that the receive FIFO is three bytes deep. On receiving the interrupt on the first receive character, the CPU must enable the DMA controller within three received byte times to avoid receiver over-run condition. In the application example, at 400K baud, the CPU had approximately 60 μs to enable the DMA controller to avoid receiver buffer overflow. However, at higher baud rates, the CPU may not have enough time to enable the DMA controller in time. For example, at 1M baud, the CPU should enable the DMA controller within approximately 24 μs to avoid receiver buffer overrun. In most applications, this is not sufficient time. To solve this problem, the DMA controller should be left enabled before getting the interrupt on the first receive character (which is accompanied by the Rx DMA request for the appropriate channel). This will allow the DMA controller to start DMA transfer as soon as the Rx DMA request becomes active without giving the CPU enough time to re-

spond to the interrupt on the first receive character. The CPU will respond to the interrupt *after* the DMA transfer has been completed and will find the 8259A (See Application Example) responding with interrupt level 7, the lowest priority level. Note that the 8274 interrupt request was satisfied by the DMA controller, hence the interrupt on the first receive character was cleared and the 8259A had no pending interrupt. Because of no pending interrupt, the 8259A returned interrupt level 7 in response to the INTA sequence from the CPU. The user software should take care of this interrupt.

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit underrun/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI Command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

APPENDIX A
APPLICATION EXAMPLE: SOFTWARE LISTINGS

```
PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8274_S
OBJECT MODULE PLACED IN :F1:SINI74.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI74.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

                /************************************************/
                /*                                              */
                /*      INITIALIZE THE 8274 FOR SDLC MODE       */
                /*                                              */
                /*      1. RESET CHANNEL                        */
                /*      2. EXTERNAL INTERRUPTS ENABLED          */
                /*      3. NO WAIT                              */
                /*      4. PIN 10 = RTS                         */
                /*      5. NON-VECTORED INTERRUPT-8086 MODE     */
                /*      6. CHANNEL A DMA, CH B INT              */
                /*      7. TX AND RX = 8 BITS/CHAR              */
                /*      9. ADDRESS SEARCH MODE                  */
                /*      10. CD AND CTS AUTO ENABLE              */
                /*      11. X1 CLOCK                            */
                /*      12. NO PARITY                           */
                /*      13. SDLC/HDLC MODE                      */
                /*      14. RTS AND DTR                         */
                /*      15. CCITT - CRC                         */
                /*      16. TRANSMITTER AND RECEIVER ENABLED    */
                /*      17. 7EH = FLAG                          */
                /*                                              */
                /************************************************/

   1            INIT_8274_S: DO;

                $INCLUDE (:F1:PORTS.PLM)

            =   /**********************************************/
            =   /*                                            */
            =   /*      ISBC 88/45 PORT ASSIGNMENTS           */
            =   /*                                            */
            =   /**********************************************/

   2   1    =   DECLARE LIT LITERALLY 'LITERALLY';

            =   /* 8237A-5  PORTS */

   3   1    =   DECLARE CH0_ADDR          LIT     '080H',
            =           CH0_COUNT         LIT     '081H',
            =           CH1_ADDR          LIT     '082H',
            =           CH1_COUNT         LIT     '083H',
            =           CH2_ADDR          LIT     '084H',
            =           CH2_COUNT         LIT     '085H',
            =           CH3_ADDR          LIT     '086H',
            =           CH3_COUNT         LIT     '087H',
            =           STATUS_37         LIT     '088H',
            =           COMMAND_37        LIT     '088H',
            =           REQUEST_REG_37    LIT     '089H',
            =           SINGLE_MASK       LIT     '08AH',
            =           MODE_REG_37       LIT     '08BH',
PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

            =           CLR_BYTE_PTR_37 LIT       '08CH',
            =           TEMP_REG_37       LIT     '08DH',
            =           MASTER_CLEAR_37 LIT       '08DH',
            =           ALL_MASK_37       LIT     '08FH';

            =   /* 8254-2 PORTS */

   4   1    =   DECLARE CTR_00            LIT     '090H',
            =           CTR_01            LIT     '091H',
            =           CTR_02            LIT     '092H',
```

```
         =           CONTROL0_54     LIT     '093H',
         =           STATUS0_54      LIT     '093H',
         =           CTR_10          LIT     '098H',
         =           CTR_11          LIT     '099H',
         =           CTR12           LIT     '09AH',
         =           CONTROL1_54     LIT     '09BH',
         =           STATUS1_54      LIT     '09BH';

         =   /* 8255 PORTS */

 5   1   =   DECLARE PORTA_55        LIT     '0A0H',
         =           PORTB_55        LIT     '0A1H',
         =           PORTC_55        LIT     '0A2H',
         =           CONTROL_55      LIT     '0A3H';

         =   /* 8274 PORTS */

 6   1   =   DECLARE DATA_A_74       LIT     '0D0H',
         =           DATA_B_74       LIT     '0D1H',
         =           STATUS_A_74     LIT     '0D2H',
         =           COMMAND_A_74    LIT     '0D2H',
         =           STATUS_B_74     LIT     '0D3H',
         =           COMMAND_B_74    LIT     '0D3H';

         =   /* 8259A PORTS */

 7   1   =   DECLARE STATUS_POLL_59  LIT     '0E0H',
         =           ICW1_59         LIT     '0E0H',
         =           OCW2_59         LIT     '0E0H',
         =           OCW3_59         LIT     '0E0H',
         =           OCW1_59         LIT     '0E1H',
         =           ICW2_59         LIT     '0E1H',
         =           ICW3_59         LIT     '0E1H',
         =           ICW4_59         LIT     '0E1H';

         =   /* 8274 REGISTER BIT ASSIGNMENTS */
         =   /* READ REGISTER 0 */

 8   1   =   DECLARE RX_AVAIL        LIT     '01H',
         =           INT_PENDING     LIT     '02H',
         =           TX_EMPTY        LIT     '04H',
         =           CARRIER_DETECT  LIT     '08H',
         =           SYNC_HUNT       LIT     '10H',
         =           CLEAR_TO_SEND   LIT     '20H',

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

         =           END_OF_TX_MESSAGE LIT   '40H',
         =           BREAK_ABORT     LIT     '80H';

         =   /* READ REGISTER 1 */

 9   1   =   DECLARE ALL_SENT        LIT     '01H',
         =           PARITY_ERROR    LIT     '10H',
         =           RX_OVERRUN      LIT     '20H',
         =           CRC_ERROR       LIT     '40H',
         =           END_OF_FRAME    LIT     '80H';

         =   /* READ REGISTER 2 */

10   1   =   DECLARE TX_B_EMPTY      LIT     '00H',
         =           EXT_B_CHANGE    LIT     '01H',
         =           RX_B_AVAIL      LIT     '02H',
         =           RX_B_SPECIAL    LIT     '03H',
         =           TX_A_EMPTY      LIT     '04H',
         =           EXT_A_CHANGE    LIT     '05H',
         =           RX_A_AVAIL      LIT     '06H',
         =           RX_A_SPECIAL    LIT     '07H';
```

```
              =   /* 8237 BIT ASSIGNMENTS */

  11    1     =   DECLARE CH0_SEL          LIT      '00H',
              =           CH1_SEL          LIT      '01H',
              =           CH2_SEL          LIT      '02H',
              =           CH3_SEL          LIT      '03H',
              =           WRITE_XFER       LIT      '04H',
              =           READ_XFER        LIT      '08H',
              =           DEMAND_MODE      LIT      '00H',
              =           SINGLE_MODE      LIT      '40H',
              =           BLOCK_MODE       LIT      '80H',
              =           SET_MASK         LIT      '04H';

  12    1         DELAY_S: PROCEDURE PUBLIC;
  13    2         DECLARE D WORD;
  14    2         D=0;
  15    2         DO WHILE D<800H;
  16    3         D=D+1;
  17    3         END;
  18    2         END DELAY_S;

  19    1         INIT_8274_SDLC_S:    PROCEDURE PUBLIC;

  20    2         DECLARE C    BYTE;

                  $EJECT


PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

                 /* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B */

                 /*  FORMAT IS: WRITE REGISTER, REGISTER DATA  */
                 /*       INITIALIZE CHANNEL ONLY                      */

  21    2        DECLARE TABLE_74_A(*) BYTE DATA
                         (00H,18H,        /* CHANNEL RESET */
                         00H,80H,         /* RESET TX CRC */
                         02H,11H,         /* PIN 10=RTSB, A DMA, B INT */
                         04H,20H,         /* SDLC/HDLC MODE, NO PARITY */
                         07H,07EH,        /* SDLC FLAG */
                         01H,08H,         /* RX DMA ENABLE */
                         05H,0EBH,        /* DTR, RTS, 8 TX BITS, TX ENABLE, TX CRC ENABLE */
                         06H,55H,         /* DEFAULT ADDRESS */
                         03H,0D9H,        /* 8 RX BITS, AUTO ENABLES, HUNT MODE, */
                                          /* RX CRC ENABLE */
                         0FFH);           /* END OF INITIALIZATION TABLE */

  22    2        DECLARE TABLE_74_B(*) BYTE DATA
                         (02H,00H,        /* INTERRUPT VECTOR */
                         01H,1CH,         /* STATUS AFFECTS VECTOR */
                         0FFH);           /* END */

                 /* INITIALIZE THE 8254 */

  23    2        OUTPUT(CONTROL0_54)=36H;
  24    2        OUTPUT(CTR_00) = LOW(20);       /* BAUD RATE = 400K_BAUD*/
  25    2        OUTPUT(CTR_00) = HIGH(20);      /* BAUD RATE = 400K_BAUD*/

                 /* INITIALIZE THE 8274 */

  26    2        C=0;
  27    2        DO WHILE TABLE_74_B(C) <> 0FFH;
  28    3            OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
  29    3            C=C+1;
  30    3            OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
  31    3            C=C+1;
  32    3        END;
```

REFERENCES

1. *IBM Document No. GA27-3004-2: General Information — Binary Synchronous Communications*
2. *Application Note AP134: Asynchronous Communication with the 8274 Multiple Protocol Serial Controller.* Intel Corp., Ca.
3. *8274 MPSC Data Sheet*, Intel Corporation, Ca.
4. *iSBC 88/45 Hardware Reference Manual*, Intel Corp., Ca.
5. *Computer Networks and Distributed Processing by James Martin.* Prentice Hall, Inc., N.J.

```
 33   2      C=0;
 34   2      DO WHILE TABLE_74_A(C) <> OFFH;
 35   3          OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
 36   3          C=C+1;
 37   3          OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
 38   3          C=C+1;
 39   3      END;
 40   2          CALL     DELAY_S;

 41   2      RETURN;
 42   2      END INIT_8274_SDLC_S;
 43   1      END INIT_8274_S;
```

```
PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


MODULE INFORMATION:

      CODE AREA SIZE     = 00A8H    168D
      CONSTANT AREA SIZE = 0000H      0D
      VARIABLE AREA SIZE = 0003H      3D
      MAXIMUM STACK SIZE = 0006H      6D
      213 LINES READ
      0 PROGRAM WARNINGS
      0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8237_CHA
OBJECT MODULE PLACED IN :F1:SINI37.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI37.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC
TEST) COMPACT NOINTVECTOR ROM

             /***********************************************************************/
             /*                                                                     */
             /*      8237    INITIALIZATION ROUTINE  FOR DMA TRANSFER               */
             /*                                                                     */
             /***********************************************************************/

  1          INIT_8237_CHA: DO;

             $NOLIST

 12   1      INIT_8237_S: PROCEDURE PUBLIC;


 13   2      OUTPUT(MASTER_CLEAR_37)=0;
 14   2      OUTPUT(COMMAND_37) = 20H;          /* EXTENDED WRITE */
 15   2      OUTPUT(ALL_MASK_37) = 0FH;         /* MASK ALL REQUESTS */
 16   2      OUTPUT(MODE_REG_37) = (SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
 17   2      OUTPUT(MODE_REG_37) = (SINGLE_MODE OR READ_XFER OR CH1_SEL);
 18   2      OUTPUT(CLR_BYTE_PTR_37) = 0;
 19   2      OUTPUT(CH0_ADDR) = 00;        /* RECEIVE BUFF AT 900H */
 20   2      OUTPUT(CH0_ADDR) = 09H;
 21   2      OUTPUT(CH0_COUNT) = 0H;
 22   2      OUTPUT(CH0_COUNT) = 01;
 23   2      OUTPUT(CH1_ADDR) = 00;        /* TRANSMIT BUFF AT 800H */
 24   2      OUTPUT(CH1_ADDR) = 08H;
 25   2      OUTPUT(CH1_COUNT) = 010H;
 26   2      OUTPUT(CH1_COUNT) = 00H;
```

```
                /* ENABLE TRANSFER */
27    2         OUTPUT(SINGLE_MASK) = CH1_SEL;     /* ENABLE TX DMA */
28    2         RETURN;

29    2         END INIT_8237_S;

                /* TURN OFF THE 8237 CHANNELS 0 AND 1 */

30    1         STOP_8237_S: PROCEDURE PUBLIC;
31    2         OUTPUT(SINGLE_MASK) = CH1_SEL OR SET_MASK;
32    2         OUTPUT(SINGLE_MASK) = CH0_SEL OR SET_MASK;
33    2         RETURN;
34    2         END STOP_8237_S;
35    1         END INIT_8237_CHA;


MODULE INFORMATION:

     CODE AREA SIZE     = 004CH      76D
     CONSTANT AREA SIZE = 0000H       0D
     VARIABLE AREA SIZE = 0000H       0D
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


     MAXIMUM STACK SIZE = 0002H       2D
     163 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INTR_8274_S
OBJECT MODULE PLACED IN :F1:SINTR.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINTR.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

                /******************************************/
                /*                                        */
                /*      8274 INTERRUPT ROUTINE            */
                /*                                        */
                /******************************************/

 1              INTR_8274_S: DO;
                $NOLIST
12    1         DECLARE TEMP BYTE;
13    1         DECLARE (RESULTS_S, TXDONE_S, RXDONE_S) BYTE EXTERNAL;
14    1         DECLARE INT_VEC POINTER AT (140);
15    1         DECLARE INT_VEC_STORE POINTER;
16    1         DECLARE MASK_59 BYTE;
17    1         DECLARE DONE            LIT     'OFFH',
                        NOT_DONE        LIT     'OOH',
                        PASS            LIT     'OFFH',
                        FAIL            LIT     'OOH';

                /****************************/
                /* IGNORE INTERRUPT HANDLER */
                /****************************/

18    1         IGNORE_INT: PROCEDURE;

19    2         RESULTS_S = FAIL;
20    2         RETURN;
21    2         END IGNORE_INT;
```

```
              /***************************************************/
              /* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
              /***************************************************/

  22    1     CHA_EXTERNAL_CHANGE: PROCEDURE;

  23    2     TEMP = INPUT(STATUS_A_74);          /* STATUS REG 1*/
  24    2     IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
  25    2          TXDONE_S=DONE;
  26    2     ELSE DO;
  27    3          TXDONE_S=DONE;
  28    3          RESULTS_S=FAIL;
  29    3           END;
  30    2     OUTPUT(COMMAND_A_74) = 10H;  /* RESET EXT/STATUS INTERRUPTS */
  31    2     RETURN;
  32    2     END CHA_EXTERNAL_CHANGE;

              $EJECT
```

PL/M-86 COMPILER   iSBC 88/45 8274 CHANNEL A SDLC TEST

```
              /*******************************************************/
              /* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
              /*******************************************************/

  33    1     CHA_RX_SPECIAL: PROCEDURE;

  34    2          OUTPUT(COMMAND_A_74) = 1;
  35    2          TEMP = INPUT(STATUS_A_74);
  36    2          IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
  37    2                    DO;
  38    3                      IF(TEMP AND 040H) = 040H THEN
  39    3                        RESULTS_S = FAIL;          /* CRC ERROR */
  40    3                      RXDONE_S = DONE;
  41    3                      OUTPUT(COMMAND_A_74) = 30H;  /*ERROR RESET*/
  42    3                    END;
  43    2                    ELSE DO;
  44    3                      IF (TEMP AND 20H) = 20H THEN DO;
  46    4                      RESULTS_S = FAIL;            /* RX OVERRUN ERROR*/
  47    4                      RXDONE_S = DONE;
  48    4                      OUTPUT(COMMAND_A_74) = 30H;  /*ERROR RESET*/
  49    4                      END;
  50    3                    END;
  51    2     RETURN;
  52    2     END CHA_RX_SPECIAL;


              /****************************************/
              /* CHANNEL A RECEIVE CHARACTER AVAILABLE */
              /****************************************/

  53    1     CHA_RX_CHAR: PROCEDURE;
  54    2     OUTPUT(SINGLE_MASK) = CH0_SEL;           /*ENABLE RX DMA CHANNEL*/
  55    2     RETURN;
  56    2     END CHA_RX_CHAR;

              $EJECT
```

PL/M-86 COMPILER   iSBC 88/45 8274 CHANNEL A SDLC TEST

```
              /* ENABLE 8274 INTERRUPTS - SET UP THE 8259A */

  57    1     ENABLE_INTERRUPTS_S: PROCEDURE PUBLIC;

  58    2     DECLARE CHA_INT_ON LIT '0F7H';

  59    2     DISABLE;

  60    2     CALL     SET$INTERRUPT(39, INT_39);
```

```
61   2      INT_VEC_STORE = INT_VEC;
62   2      INT_VEC = INTERRUPT$PTR(INT_8274_S);
63   2      MASK_59 = INPUT(OCW1_59);

64   2      OUTPUT(OCW1_59) = MASK_59 AND CHA_INT_ON;

65   2      RETURN;
66   2      END ENABLE_INTERRUPTS_S;

            /* DISABLE 8274 INTERRUPTS - SET UP THE 8259A */

67   1      DISABLE_INTERRUPTS_S: PROCEDURE PUBLIC;

68   2      DISABLE;

69   2      INT_VEC = INT_VEC_STORE;

70   2      OUTPUT(OCW1_59) = MASK_59;
71   2      ENABLE;

72   2      RETURN;
73   2      END DISABLE_INTERRUPTS_S;

            /* CHANNEL B RECEIVE CHARACTER AVAILABLE */

74   1      CHB_RX_CHAR: PROCEDURE;

75   2      TEMP=INPUT(DATA_B_74);

76   2      OUTPUT(COMMAND_B_74) = 38H;
77   2      RETURN;
78   2      END CHB_RX_CHAR;

            $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


            /**************************/
            /* MAIN INTERRUPT ROUTINE */
            /**************************/

79   1      INT_8274_S: PROCEDURE INTERRUPT 35 PUBLIC;

80   2      OUTPUT(COMMAND_B_74) = 2;                   /* SET POINTER TO 2*/
81   2      TEMP = INPUT(STATUS_B_74) AND 07H;          /* READ INTERRUPT VECTOR */
                                                        /* CHECK FOR CHA INT ONLY*/
            /* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
82   2      DO CASE TEMP;
83   3          CALL    IGNORE_INT;                 /* V2V1V0 = 000*/
84   3          CALL    IGNORE_INT;                 /* V2V1V0 = 001*/
85   3          CALL    CHB_RX_CHAR;                /* V2V1V0 = 010*/
86   3          CALL    IGNORE_INT;                 /* V2V1V0 = 011*/
87   3          CALL    IGNORE_INT;                 /* V2V1V0 = 100*/
88   3          CALL    CHA_EXTERNAL_CHANGE;        /* V2V1V0 = 101*/
89   3          CALL    CHA_RX_CHAR;                /* V2V1V0 = 110*/
90   3          CALL    CHA_RX_SPECIAL;             /* V2V1V0 = 111*/
91   3      END;
92   2      OUTPUT(COMMAND_A_74) =38H;   /* END OF INTERRUPT FOR 8274 */
93   2      OUTPUT(OCW2_59) = 63H;       /* 8259 EOI */
94   2      OUTPUT(OCW1_59) = INPUT(OCW1_59) AND OF7H;
95   2      RETURN;
96   2      END INT_8274_S;


            /* DEFAULT INTERRUPT ROUTINE - 8259A INTERRUPT 7 */
            /* REQUIRED ONLY WHEN DMA CONTROLLER IS ENABLED  */
            /* BEFORE RECEIVING FIRST CHARACTER WHICH IS     */
            /* AT HIGH BAUD RATES  LIKE 800K BAUD. READ APP. */
            /* NOTE SECTION 6 FOR DETAILS                    */
```

```
  97   1        INT_39: PROCEDURE INTERRUPT 39;
  98   2                OUTPUT(OCW2_59) = 20H;      /* NON-SPECIFIC EOI */
  99   2                OUTPUT(OCW1_59) = INPUT(OCW1_59) AND OF7H;
 100   2                RESULTS_S = FAIL;
 101   2        END INT_39;

 102   1        END INTR_8274_S;


 MODULE INFORMATION:

      CODE AREA SIZE     = 01BFH    447D
      CONSTANT AREA SIZE = 0000H      0D
      VARIABLE AREA SIZE = 0006H      6D
      MAXIMUM STACK SIZE = 0022H     34D
      295 LINES READ
      0 PROGRAM WARNINGS
      0 PROGRAM ERRORS

 END OF PL/M-86 COMPILATION

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE STEST
OBJECT MODULE PLACED IN :F1:STEST.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:STEST.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)
COMPACT NOINTVECTOR ROM

               /***********************************************************************/
               /*                                                                     */
               /*       iSBC 545 PORT A (8274) SDLC TEST                              */
               /*                                                                     */
               /***********************************************************************/

   1           STEST: DO;


   2    1      DELAY_S: PROCEDURE EXTERNAL;
   3    2      END DELAY_S;

   4    1      ENABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
   5    2      END ENABLE_INTERRUPTS_S;

   6    1      DISABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
   7    2      END DISABLE_INTERRUPTS_S;

   8    1      INIT_8274_SDLC_S: PROCEDURE EXTERNAL;
   9    2      END INIT_8274_SDLC_S;

  10    1      INIT_8237_S: PROCEDURE EXTERNAL;
  11    2      END INIT_8237_S;

  12    1      STOP_8237_S: PROCEDURE EXTERNAL;
  13    2      END STOP_8237_S;

  14    1      VERIFY_TRANSFER_S: PROCEDURE EXTERNAL;
  15    2      END VERIFY_TRANSFER_S;

  16    1      INT_8274_S: PROCEDURE INTERRUPT 35 EXTERNAL;
  17    2      END INT_8274_S;
               $NOLIST
               $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


  28    1      DECLARE (RESULTS_S,TXDONE_S,RXDONE_S) BYTE PUBLIC;
  29    1      DECLARE DONE              LIT      'OFFH',
                       NOT_DONE          LIT      'OOH',
                       PASS              LIT      'OFFH',
                       FAIL              LIT      'OOH';
```

```
          $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


 30   1       CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

 31   2                 CALL     ENABLE_INTERRUPTS_S;
 32   2                 CALL     INIT_8274_SDLC_S;
 33   2                 ENABLE;
 34   2                 OUTPUT(COMMAND_A_74) = 28H;  /* RESET TX INT/DMA       */
 35   2                 OUTPUT(COMMAND_B_74) = 28H;  /* BEFORE INITIALIZING 8237*/
 36   2                 CALL     INIT_8237_S;
 37   2                 OUTPUT(DATA_A_74) = 55H;   /* LOAD FIRST CHARACTER FROM CPU*/

                        /* TO ENSURE CRC TRANSMISSION RESET TX UNDERRUN LATCH*/
 38   2                 OUTPUT(COMMAND_A_74) = OCOH;
 39   2                 RXDONE_S, TXDONE_S=NOT_DONE;        /* CLEAR ALL FLAGS */
 40   2                 RESULTS_S=PASS;                     /* FLAG SET FOR MONITOR*/

 41   2                 DO WHILE TXDONE_S=NOT_DONE;         /* DO UNTIL TERMINAL COUNT*/
 42   3                 END;

 43   2                 DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
                        /* WAIT FOR CRC TO GET TRANSMITTED */
                        /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
 44   3                 END;
 45   2                 DO WHILE RXDONE_S=NOT_DONE;         /* DO UNTIL TERMINAL COUNT*/
 46   3                 END;

 47   2                 CALL     STOP_8237_S;

 48   2                 CALL     DISABLE_INTERRUPTS_S;

 49   2                 CALL     VERIFY_TRANSFER_S;

 50   2                 RETURN RESULTS_S;

 51   2       END CHA_SDLC_TEST;
 52   1       END STEST;


MODULE INFORMATION:

     CODE AREA SIZE     = 0063H      99D
     CONSTANT AREA SIZE = 0000H       0D
     VARIABLE AREA SIZE = 0003H       3D
     MAXIMUM STACK SIZE = 0004H       4D
     198 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION


PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE VECTOR_MODE
OBJECT MODULE PLACED IN :F1:VECTOR.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:VECTOR.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)


              /*********************************************************************/
              /*                                                                   */
              /*             8274 INTERRUPT HANDLING ROUTINE FOR                   */
              /*                     8274 VECTOR MODE                              */
              /*                  STATUS AFFECTS VECTOR                            */
              /*                                                                   */
              /*********************************************************************/
```

```
              /* THIS IS AN EXAMPLE OF HOW 8274 CAN BE USED IN VECTORED MODE.       */
              /* THE iSBC88/45 BOARD WAS REWIRED TO DISABLE THE PIT 8259A AND       */
              /* ENABLE THE 8274 TO PLACE ITS VECTOR ON THE DATABUS IN RESPONSE     */
              /* TO THE INTA SEQUENCE FROM THE 8088. OTHER MODIFICATIONS INCLUDED */
              /* CHANGES TO 8274 INITIALIZATION PROGRAM (SINI74) TO PROGRAM 8274   */
              /* INTO VECTORED MODE (WRITE REGISTER 2A D5=1).                       */


  1           VECTOR_MODE: DO;
              $NOLIST

 12    1      DECLARE TEMP BYTE;
 13    1      DECLARE (RESULTS_S, TXDONE, RXDONE) BYTE EXTERNAL;
 14    1      DECLARE DONE  LITERALLY 'OFFH',
                      NOT_DONE LITERALLY '00H',
                      PASS     LITERALLY 'OFFH',
                      FAIL     LITERALLY '00H';

              /**********************************************************************/
              /*      TRANSMIT INTERRUPT CHANNEL A INTERRUPT WILL NOT BE SEEN IN THE */
              /*      DMA OPERATION.                                                  */
              /**********************************************************************/

 15    1         TX_INTERRUPT_CHA: PROCEDURE INTERRUPT 84;
 16    2         OUTPUT(COMMAND_A_74) = 00101000B;          /*RESET TXINT PENDING*/
 17    2         OUTPUT(COMMAND_A_74) = 00111000B;          /*EOI*/
 18    2         END TX_INTERRUPT_CHA;


              /**********************************************************************/
              /*      EXTERNAL/STATUS INTERRUPT PROCEDURE: CHECKS FOR END OF MESSAGE */
              /*      ONLY. IF THIS IS NOT TRUE THEN THE FAIL FLAG IS SET. HOWEVER,  */
              /*      A USER PROGRAM SHOULD CHECK FOR OTHER EXT/STATUS CONDITIONS    */
              /*      ALSO IN RR1 AND THEN TAKE APPROPRIATE ACTION BASED ON THE      */
              /*      APPLICATION.                                                    */
              /**********************************************************************/

 19    1      EXT_STAT_CHANGE_CHA: PROCEDURE INTERRUPT 85;
 20    2              TEMP   = INPUT(STATUS_A_74);
 21    2              IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
 22    2                 TXDONE = DONE;
 23    2              ELSE DO;
 24    3                 TXDONE = DONE;

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


 25    3                 RESULTS_S = FAIL;
 26    3               END;

 27    2         OUTPUT(COMMAND_A_74) = 00010000B;          /*RESET EXT STAT INT*/
 28    2             OUTPUT(COMMAND_A_74) = 00111000B;         /*EOI*/
 29    2             RETURN;
 30    2      END EXT_STAT_CHANGE_CHA;

              /**********************************************************************/
              /*      RECEIVER CHARACTER AVAILABLE INTERRUPT WILL APPEAR ONLY ON FIRST*/
              /*      RECEIVE CHARACTER. SINCE DMA CONTROLLER HAS BEEN ENABLED BEFORE */
              /*      THE FIRST CHARACTER IS RECEIVED, THE RECEIVER REQUEST IS        */
              /*      SERVICED BY THE DMA CONTROLLER.                                 */
              /**********************************************************************/

 31    1      RX_CHAR_AVAILABLE_CHA: PROCEDURE INTERRUPT 86;
 32    2         OUTPUT(COMMAND_A_74) = 00111000B;       /*EOI*/
 33    2             RETURN;
 34    2      END RX_CHAR_AVAILABLE_CHA;
              $EJECT
```

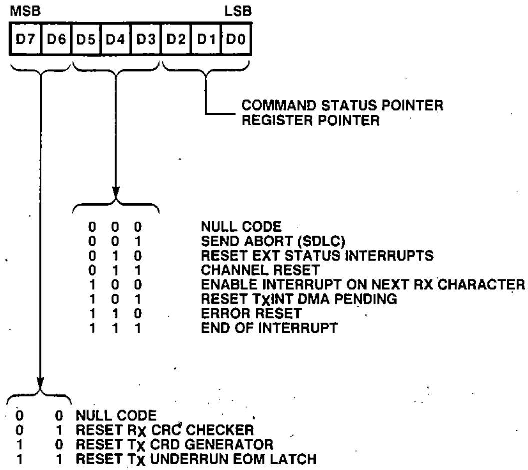
PL/M-86 COMPILER   iSBC 88/45 8274 CHANNEL A SDLC TEST

```
/************************************************************************/
/*      SPECIAL RECEIVE CONDITION INTERRUPT SERVICE ROUTINE CHECKS FOR */
/*      END OF FRAME BIT ONLY. SEE SPECIAL SERVICE ROUTINE FOR NON-    */
/*      VECTORED MODE FOR CRC CHECK AND OVERRUN ERROR CHECK.            */
/************************************************************************/

SPECIAL_RX_CONDITION_CHA: PROCEDURE INTERRUPT 87;

        OUTPUT(COMMAND_A_74) = 1;                       /*POINTER 1*/
        TEMP = INPUT(STATUS_A_74);
        IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
              RXDONE = DONE;
        ELSE DO;
              RXDONE = DONE;
              RESULTS_S = FAIL;
             END;
    OUTPUT(COMMAND_A_74) = 00110000B;           /*ERROR RESET*/
    OUTPUT(COMMAND_A_74) = 00111000B;           /*EOI*/
        RETURN;
END SPECIAL_RX_CONDITION_CHA;

ENABLE_INTERRUPTS: PROCEDURE PUBLIC;
DISABLE;
CALL SET$INTERRUPT(84, TX_INTERRUPT_CHA);
CALL SET$INTERRUPT(85, EXT_STAT_CHANGE_CHA);
CALL SET$INTERRUPT(86, RX_CHAR_AVAILABLE_CHA);
CALL SET$INTERRUPT(87, SPECIAL_RX_CONDITION_CHA);
RETURN;
END ENABLE_INTERRUPTS;

END VECTOR_MODE;
/***************************************************************************/
/***************************************************************************/
```

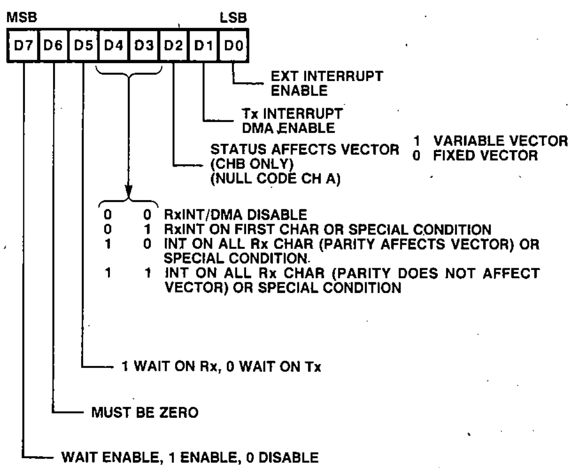
```
MODULE INFORMATION:

     CODE AREA SIZE     = 012EH    302D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 001EH     30D
     226 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

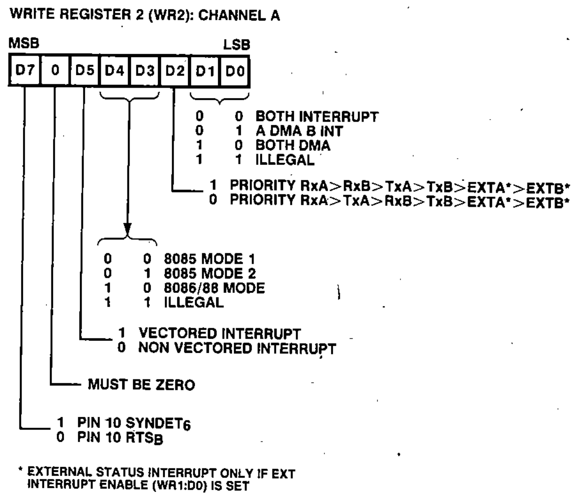
APPENDIX B
MPSC READ/WRITE REGISTER DESCRIPTIONS

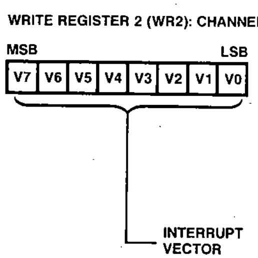
WRITE REGISTER 0 (WR0):

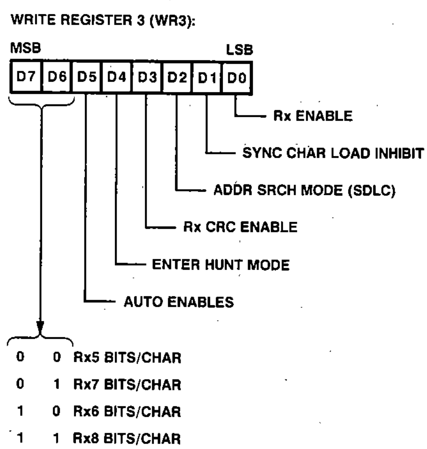


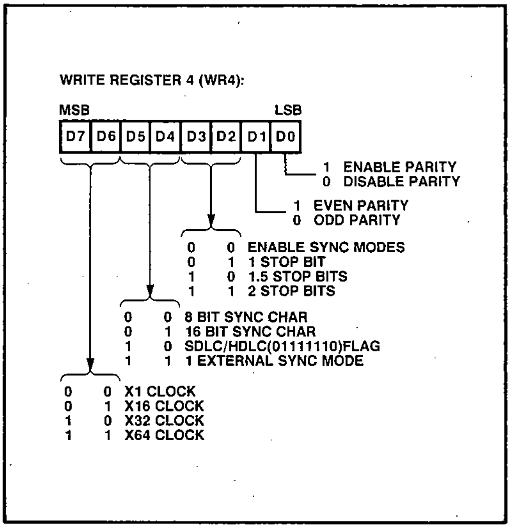
WRITE REGISTER 1 (WR1):

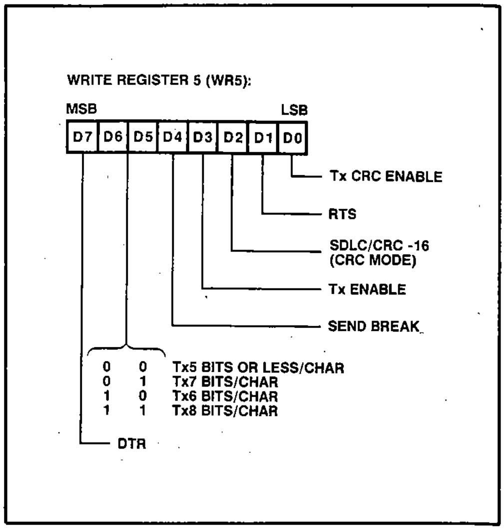


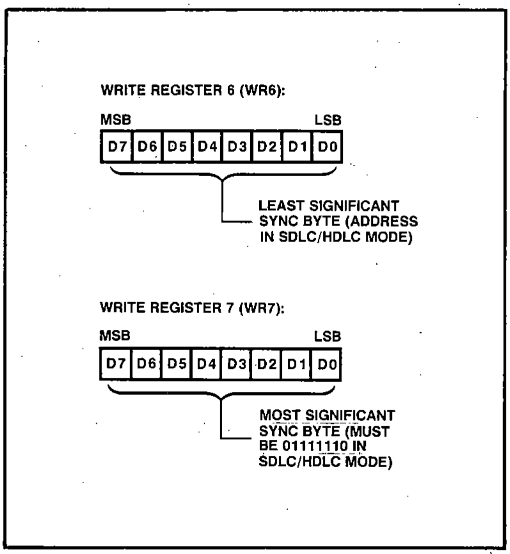

WRITE REGISTER 2 (WR2): CHANNEL A
MSB
LSB
D7 0 D5 D4 D3 D2 D1 D0
0 0 BOTH INTERRUPT
0 1 A DMA B INT
1 0 BOTH DMA
1 1 ILLEGAL
1 PRIORITY RxA>RxB>TxA>TxB>EXTA*>EXTB*
0 PRIORITY RxA>TxA>RxB>TxB>EXTA*>EXTB*
0 0 8085 MODE 1
0 1 8085 MODE 2
1 0 8086/88 MODE
1 1 ILLEGAL
1 VECTORED INTERRUPT
0 NON VECTORED INTERRUPT
MUST BE ZERO
1 PIN 10 SYNDETB
0 PIN 10 RTSB
* EXTERNAL STATUS INTERRUPT ONLY IF EXT INTERRUPT ENABLE (WR1:D0) IS SET

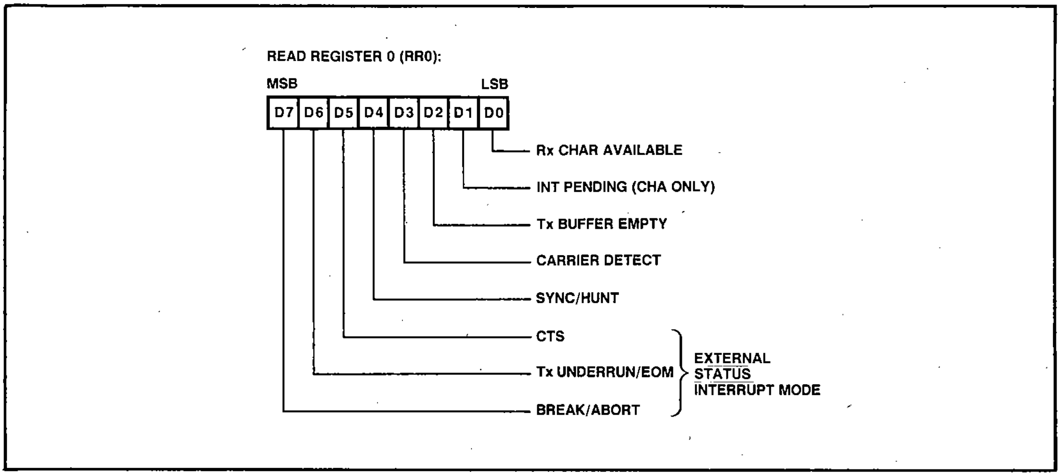

WRITE REGISTER 2 (WR2): CHANNEL B
MSB
LSB
V7 V6 V5 V4 V3 V2 V1 V0
INTERRUPT VECTOR

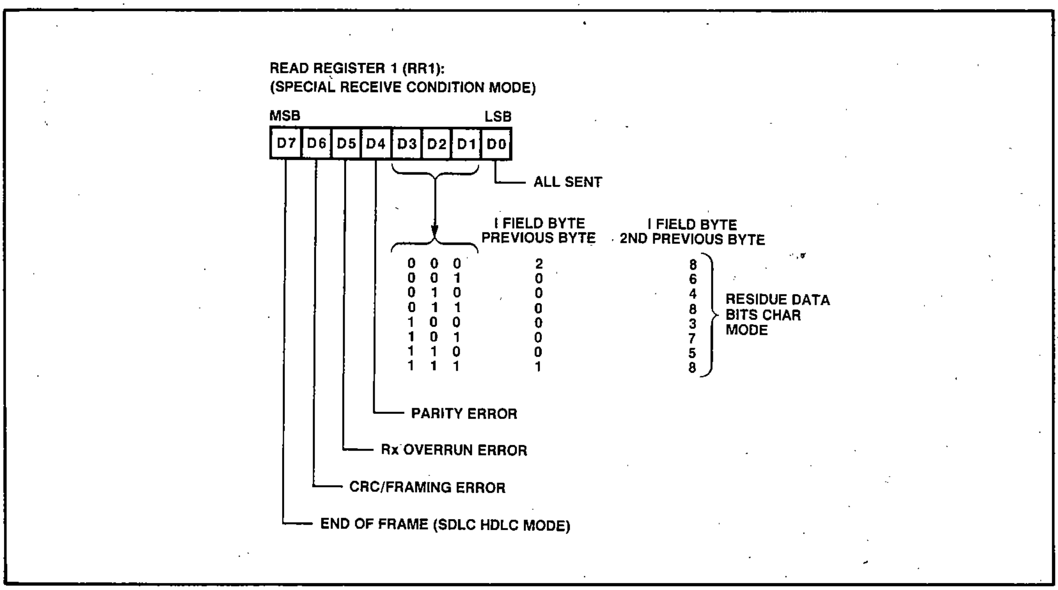

WRITE REGISTER 3 (WR3):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
Rx ENABLE
SYNC CHAR LOAD INHIBIT
ADDR SRCH MODE (SDLC)
Rx CRC ENABLE
ENTER HUNT MODE
AUTO ENABLES
0 0 Rx5 BITS/CHAR
0 1 Rx7 BITS/CHAR
1 0 Rx6 BITS/CHAR
1 1 Rx8 BITS/CHAR

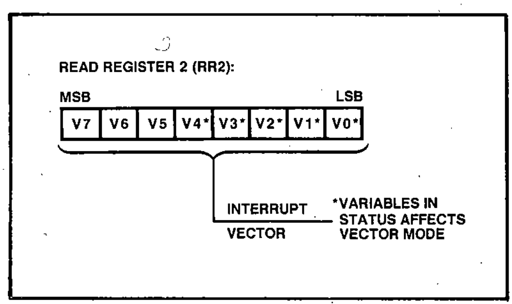

WRITE REGISTER 4 (WR4):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
1 ENABLE PARITY
0 DISABLE PARITY
1 EVEN PARITY
0 ODD PARITY
0 0 ENABLE SYNC MODES
0 1 1 STOP BIT
1 0 1.5 STOP BITS
1 1 2 STOP BITS
0 0 8 BIT SYNC CHAR
0 1 16 BIT SYNC CHAR
1 0 SDLC/HDLC(01111110)FLAG
1 1 1 EXTERNAL SYNC MODE
0 0 X1 CLOCK
0 1 X16 CLOCK
1 0 X32 CLOCK
1 1 X64 CLOCK


WRITE REGISTER 5 (WR5):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
Tx CRC ENABLE
RTS
SDLC/CRC -16 (CRC MODE)
Tx ENABLE
SEND BREAK
0 0 Tx5 BITS OR LESS/CHAR
0 1 Tx7 BITS/CHAR
1 0 Tx6 BITS/CHAR
1 1 Tx8 BITS/CHAR
DTR


WRITE REGISTER 6 (WR6):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
LEAST SIGNIFICANT SYNC BYTE (ADDRESS IN SDLC/HDLC MODE)
WRITE REGISTER 7 (WR7):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
MOST SIGNIFICANT SYNC BYTE (MUST BE 01111110 IN SDLC/HDLC MODE)


READ REGISTER 0 (RR0):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
Rx CHAR AVAILABLE
INT PENDING (CHA ONLY)
Tx BUFFER EMPTY
CARRIER DETECT
SYNC/HUNT
CTS
Tx UNDERRUN/EOM
BREAK/ABORT
EXTERNAL STATUS INTERRUPT MODE


READ REGISTER 1 (RR1):
(SPECIAL RECEIVE CONDITION MODE)
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
ALL SENT
I FIELD BYTE PREVIOUS BYTE
I FIELD BYTE 2ND PREVIOUS BYTE
0 0 0 2 8
0 0 1 0 6
0 1 0 0 4
0 1 1 0 8
1 0 0 0 3
1 0 1 0 7
1 1 0 0 5
1 1 1 1 8
RESIDUE DATA BITS CHAR MODE
PARITY ERROR
Rx OVERRUN ERROR
CRC/FRAMING ERROR
END OF FRAME (SDLC HDLC MODE)


READ REGISTER 2 (RR2):
MSB
LSB
V7 V6 V5 V4* V3* V2* V1* V0*
INTERRUPT VECTOR
*VARIABLES IN STATUS AFFECTS VECTOR MODE